УДК 681.3

В. П. Симоненко, д-р техн. наук,
А. М. Сергиенко, канд. техн. наук
Национальный технический университет Украины «КПИ»
(Украина, 03056, Киев, пр. Победы, 37,
тел.: 4549337, E-mail: aser@comsys.ntu-kpi.kiev.ua)

Алгоритмические модели обработки потоков данных

Рассмотрены свойства различных вычислительных моделей в виде графов потоков данных и предложены рекомендации по выбору модели, наиболее пригодной для проектирования конвейерных вычислителей.

Розглянуто властивості різних обчислювальних моделей у вигляді графів потоків даних і запропоновано рекомендації щодо вибору моделі, найбільш придатної для проектування конвейерних обчислювачів.

К л ю ч е в ы е с л о в а: граф потоков данных, отображение алгоритма, вычислительная система.

По общепринятому определению, алгоритм — это вычислительный процесс, имеющий начало, детерминированную последовательность действий и конец и дающий ожидаемые результаты при различных исходных данных [1]. Это определение относится к трансформирующим вычислительным системам (ВС), не имеющим ограничения на быстродействие. ВС реального времени отличаются тем, что периоды выполнения алгоритмов в них не превосходят заданных временных ограничений. Такие ВС разделяют на интерактивные, если они реагируют на входные данные с сответствующей им скоростью, и реактивные, которые не допускают потери этих данных вследствие недостаточного быстродействия [2,3]. Персональные компьютеры, устройства мобильной связи и средства цифровой обработки сигналов (ЦОС) относятся к ВС реального времени. Они и многие другие ВС выполняют периодические алгоритмы, обрабатывающие потоки данных и (или) команд.

Цифровая обработка сигналов основана на том, сигналы обрабатываются как потоки данных. Поэтому область ЦОС наиболее достоверно отображает задачи, алгоритмы и ВС, в которых наблюдаются такие потоки. На современном этапе развития ЦОС алгоритмы, обеспечивающие беспроводную связь, компрессию изображений, моделирование трехмерных

сцен, не могут быть реализованы на процессоре персонального компьютера в реальном времени. Для их реализации необходимо использовать специальные ВС на основе заказных сверхбольших интегральных схем (СБИС), многопроцессорных сигнальных микропроцессоров, а также програмируемые логические интегральные схемы (ПЛИС). На современном этапе развития ЦОС программист, разработавший новый алгоритм ЦОС, имеет возможность немедленно внедрить его, но этого не произойдет, если он не владеет в совершенстве параллельным программированием [4].

Для достижения высокой производительности ВС необходимо программирование многопроцессорных параллельных систем. Однако при наличии более четырех процессоров их программирование (учитывая современное состояние технологии разработки программ) представляет собой весьма трудоемкую работу. В настоящее время широко применяется парадигма многопотоковой обработки, поддерживаемая как на уровне матобеспечения, так и на аппаратном уровне современных компьютеров. Но, как показано, например, в [5], многопотоковая вычислительная модель не гарантирована от блокировок, и при переходе к многоядерным микропроцессорам вероятность блокировок возрастает. Поэтому необходимо использование других парадигм решения задач обработки потоков данных.

Новые задачи, возникающие в области развития ЦОС, не могут быть решены без разработки методов и средств отображения алгоритмов в СБИС и ПЛИС, программирования многопроцессорных ВС. Такая разработка возможна, прежде всего, при условии нового видения природы алгоритмов обработки потоков данных. К сожалению, в отечественной литературе не опубликовано систематической информации об алгоритмических моделях обработки потоков данных и об их применении в практике программирования и синтеза параллельных ВС. Предлагаемый обзор существующих моделей построения периодических алгоритмов обработки потоков данных отображает состояние научной мысли в этой области.

Алгоритм и вычислительная модель. По определению Поста и Тьюринга, алгоритм — это вычислительный процесс, выполняемый некоторой моделью вычислителя, которая сконструирована в рамках точных математических понятий [6]. Такое определение алгоритма дает возможность, во-первых, классифицировать алгоритмы по типу модели вычислителя, во-вторых, сравнивать между собой различные алгоритмы путем их эквивалентного преобразования в алгоритмы, заданные на стандартной модели вычислителя, такой как машина Тьюринга, в-третьих, конструировать модели вычислителей, наиболее пригодные для эффективного задания того или иного класса алгоритмов.

Определение алгоритма, приведенное в начале статьи относится, прежде всего, к неймановской модели вычислителя, которая последова-

тельно выполняет в строгом порядке команды, изменяющие состояние памяти вычислителя от начального до конечного. На такой модели выполняются операторные, императивные алгоритмы.

В функциональных алгоритмах вычисления представляют собой множество тождеств функций. При этом результаты функций вычисляются только тогда, когда для них существуют сответствующие входные данные. В функциональном алгоритме нет ограничений на порядок вычисления функций кроме порядка, заданного зависимостями по данным или управлению.

Модель вычислителя, реализующую функциональный алгоритм, принято представлять в виде графа. В такой модели вершины графа означают функции или операторы, или локальные процессы алгоритма, а дуги — каналы передачи данных, зависимости по данным и управлению. Реализация алгоритма на такой модели представляет собой передачу данных в направлении дуг и обход вершин в сответствии с их инцидентностью, наличием данных на их входах или по другим правилам. В отличие от императивных алгоритмов здесь порядок выполнения операторов нестрогий, а параллелизм алгоритма задан явным образом.

Рассмотрим несколько известных графовых моделей функциональных алгоритмов и их основные свойства.

Сеть Петри — наиболее известная графовая модель вычислений. Граф этой сети состоит из вершин-позиций и вершин-переходов, попарно связанных дугами. Выполнение алгоритма в этой модели осуществляется с помощью меток в вершинах-позициях, которые асоциируются с даными. Если меток на входах вершины-перехода достаточно, то она срабатывает и сответствующая метка появляется на ее выходах. Выполнение алгоритма состоит из серии срабатываний переходов в произвольном порядке, пока не останется условий для таких событий [7].

Сети Петри используются, прежде всего, для моделирования работы асинхронных систем, например протоколов обмена данными [8]. Существует достаточное число работ [9—12], посвященных применению этих сетей в проектировании вычислительной аппаратуры, но они не получили широкого распространения. Основной причиной этого является сложность анализа сети Петри, определения ее безопасности, отсутствия блокировок.

Граф потоков данных (ГПД) — направленный граф, вершины которого — акторы — представляют операции, а дуги — каналы передачи данных. Акторы используют данные со своих входов, называемые метками или токенами, и выдают метки на свои выходы. ГПД, предложенный Деннисом, представляет собой модель, в которой вычисления определяются наличием данных на информационных и управляющих дугах [13].

В отличие от неймановского процессора, в котором вычисления управляются потоком команд, задаваемым счетчиком команд, в компьютере потоков данных, предложенном Деннисом, управление сводится к слежению за наличием готовых данных и направлению их на входы акторов.

Граф Карпа и Миллера (граф вычислений) — один из первых ГПД для задания алгоритмов обработки потоков данных [13]. В нем k-й дуге, представляющей очередь, присвоено число меток в начале вычислений, число меток, входящих в очередь при выполнении оператора, число меток W_k, выходящих из очереди при выполнении актора, и минимальная длина очереди T_k, необходимая для запуска актора. При выполнении вычислительного процесса акторы запускаются, если $W_k \geq T_k$, и из потоков читаются W_k меток. Между графом Карпа и Миллера и сетью Петри может быть взаимно-однозначная связь — очереди данных соответствуют вершинам-позициям, а акторы — вершинам-переходам [14].

Сеть процессов Кана — вычислительная модель, в которой асинхронные процессы взаимодействуют между собой через последовательные буферы данных, называемые потоками. Эти потоки подчиняются дисциплине: первый вошел в буфер —первый вышел, т.е. FIFO [13, 15]. В этой модели вершина процесса может выполнять ввод данных в произвольном порядке. Но после считывания данного процесс блокируется, пока не поступит новое данное, т.е. процесс запускается по оператору wait on data, а само данное удаляется из буфера. В общем случае, если глубина буферов FIFO ограничена, эта сеть может быть заблокирована, причем невозможно определить заранее, совершится ли такая блокировка. Поэтому корректное выполнение алгоритма может быть гарантировано лишь при динамическом составлении расписания и потенциально неограниченной глубине буфера FIFO.

В отличие от сети Петри в сети процессов Кана состояние данных в потоках не зависит от расписания выполнения процессов, т.е. потоки данных в модели детерминированные [13]. Эта сеть явилась стимулом для появления многих других моделей обработки потоков данных. Подробный обзор моделей потоков данных и библиографические источники, посвященные изучению свойств этих моделей, приведены в [13, 15]. Для классификации моделей потоков данных рассмотрим определение потока данных и его параметры.

Классификация моделей обработки потоков данных по типу потока данных. Все модели вычислителей потоков данных имеют общую особенность, состоящую в том, что их компоненты — узлы, операторы, акторы или процессы — соединены между собой потоками данных. Поток данных — это средство передачи конечного или, в предельном случае,

бесконечного количества упорядоченных данных между компонентами вычислительной модели. Данные в потоке, упорядоченные по номерам ассоциированных с ними тегов, например временных меток, принято называть сигналом.

Вычислительные модели различаются между собой способом выполнения потоков данных, который зависит от определенных свойств или признаков потока, т.е. от следующих альтернатив:

поток однонаправленный или двунаправленный;

у потока один источник данных или их несколько;

с потока считывает данные один процесс или несколько процессов;

как в потоке данные различаются по времени возникновения, использования;

как в потоке данные различаются между собой порядком (например, в буфере FIFO);

разрешено ли в потоке данные дублировать или перезаписывать;

разрешена ли проверка наличия данных в потоке;

рассматривается в модели время как беспрерывная сущность, когда сигнал есть функция от времени, или как дискретные события, когда отсчет сигнала представлен парой момент времени (номер такта) — значение, или процессы вообще развиваются безотносительно времени, когда сигнал — это упорядоченное множество отсчетов;

равно ли количество переданных данных в поток за определенный промежуток времени количеству использованных данных или это количество изменяется динамически;

обращаются ли одновременно к потоку источник и приемник данного [15].

Синхронные и асинхронные модели. Три последних свойства потока данных, приведенных выше, характеризуют его существование во времени. Различают синхронные и асинхронные потоки и сответственно синхронные и асинхронные модели вычислений. В асинхронном потоке момент передачи данного произвольный и поток допускает произвольную задержку между загрузкой данного в него и его использованием. Это дает возможность любым способом выбирать моменты загрузки и использования данных или срабатывания сответствующих акторов при составлении расписания выполнения алгоритма. При этом количество данных в потоке в каждый момент времени может быть произвольным.

В синхронном потоке момент передачи данного или группы данных связан с некоторым событием, например сигналом синхросерии, т.е. этот момент может определяться дискретно. Если такой поток не имеет буферной памяти, то загруженное в него данное должно быть использовано

сразу или с задержкой, не превосходящей параметр синхросигнала. Метки потоков данных могут иметь временные теги их возникновения, в соответствии с которыми они являются глобально или локально упорядоченными. Метки или сигналы считаются синхронными, если они имеют теги со взаимно-однозначным соответствием. Модель считается синхронной, если все сигналы в ней — синхронные. Даже без наличия явной синхронизации в синхронной модели акторы могут использовать метки в сответствии с возрастающим порядком их временных тегов [15].

Синхронизм потока может осуществляться с помощью логических условий, например условия фронта сигнала синхросерии (это особенный случай асинхронного потока). Анализ моделей с синхронными потоками и составление для них расписания существенно упрощаются по сравнению с другими моделями.

Модели с синхронизацией обращений к потоку разделяются на модели с синхронизацией по синхросерии и модели с рандеву. В последних источник и приемник обращаются к потоку одновременно через механизм синхронизации с квитированием. Языки синхронного програмирования, такие как Esterel, Lustre, Signal, основаны на синхронизации по синхросерии, а язык Communicating Sequential Processes (CSP) и производный от него язык Occam — на синхронизации с квитированием.

В моделях с несколькими источниками и приемниками данных для согласованности доступа к разделяемой переменной чаще всего используется стратегия чтения — модификации — записи, в сответствии с которой запрещается доступ к переменной от других акторов, если один из акторов выполняет чтение переменной из потока с последующей записью в поток. Если эта или аналогичная стратегия не используется, то происходит несинхронизированная передача данных. Такая передача не гарантирует, что приемник будет читать корректное данное, записанное источником.

Модель сети процессов Кана абстрагирована от времени и поэтому является потенциально асинхронной. Вследствие этого она может задавать алгоритмы на более высоком уровне абстракции и быть трансформированной в асинхронную или синхронную модель. Кроме того, эта модель имеет потоки в виде очередей. Поэтому она может задавать алгоритмы, в которых количество данных в очереди изменяется динамически. Например, сеть процессов Кана позволяет динамически изменять количество данных, переданных в поток, вследствие чего глубина буфера потока может быть неограниченной.

Таким образом, различные модели обработки потоков данных можно классифицировать, прежде всего, по признакам их потоков, а также по строению их процессов, акторов, операторов, типу расписания.

Классификация моделей обработки потоков данных по типу расписания. Корректность задания алгоритма на модели вычислителя может быть проверена аналитически (если это возможно) или составлением расписания выполнения вычислительного процесса на данной модели. Расписание составляется при непосредственном выполнении заданного алгоритма в определенной ВС. Если структура ВС соответствует графу модели, то составление расписания состоит в определении порядка выполнения операторов (акторов, процессов) и в вычислении момента выполнения каждого оператора. Если структура ВС произвольная, то предварительно выполняют этап назначения операторов на процессоры этой ВС.

Разразличают полностью статическое, частично статическое, полностью динамическое расписание, а также расписание со статическим назначением [15]. В первом случае все этапы составления расписания выполняют до выполнения алгоритма в ВС, т.е. во время компиляции. Во втором случае назначение операторов на ресурсы и нахождение порядка их выполнения реализуются во время компиляции, а собственно момент выполнения определяется динамически. В третьем случае все этапы составления расписания выполняются динамически. Четвертый случай отличается от третьего лишь тем, что назначение на ресурсы выполняется во время компиляции и не касается вычислительных моделей, в которых эти ресурсы уже назначены.

В соответствии с первыми тремя случаями можно классифицировать алгоритмы и вычислительные модели как такие, которые имеют полностью статическое, частично статическое и полностью динамическое расписание.

При синтезе вычислителей на основе СБИС и ПЛИС соответствующие системы автоматического проектирования применяют для алгоритмов только полностью статическое расписание и очень редко частично статическое, как, например, при синтезе самосинхронизирующихся ВС. Это связано с тем, что результирующая ВС на уровне логических схем должна быть полностью детерминирована с точностью до логического элемента и, как правило, не допускает своего динамического перенастраивания во время выполнения заданного алгоритма [10, 15]. Поэтому далее будем отдельно рассматривать такие модели, которые поддерживают статическое расписание, или такие, поведение которых можно предвидеть на стадии компиляции. Последние часто называют моделями с квазистатическим расписанием.

Классификация моделей обработки потоков данных по типам акторов. Актор, или процесс, или просто оператор, обозначенный вершиной графа, выполняет некоторую функцию при срабатывании. Акторы

различают по условиям срабатывания, т.е. по функциям запуска, количеству меток, используемых и генерируемых ими, и по их внутреннему строению.

Если актор использует и воспроизводит некоторое качество меток, это количество стабильно при выполнении алгоритма и может быть определено во время компиляции, то такой актор является регулярным. Если это количество не равно константе и может изменяться во время выполнения алгоритма в зависимости от данных в потоках, то это динамический актор. Примером динамического актора является вершина переключателя, которая в зависимости от наличия метки на его логическом входе принимает метку на том или ином входе данных. Модели с динамическими акторами могут иметь динамическое или квазистатическое расписание.

У большинства моделей актор имеет одну функцию запуска, например функцию, которая запускает его при наличии определенного количества меток на всех его входах. В общем случае актор может иметь несколько функций запуска, различающихся шаблоном состояния входов. Если при некотором состоянии входов нашелся соответствующий шаблон, то актор срабатывает. Сеть с акторами, выполняющими некоторые монотонные функции и имеющими функции запуска общего вида, получила название сети обработки потоков данных (Dataflow Process Network) [21]. В зависимости от набора шаблонов эти акторы могут быть как регулярными, так и динамическими.

Граф Карпа и Миллера, в котором все акторы — регулярные, т.е. $W_k = T_k$, получил название графа синхронных потоков данных (ГСПД) или регулярного графа потоков данных [15]. Второе название перекликается с названием соответствующего подвида решетчатого графа алгоритма [16, 17] и поэтому используется реже. В ГСПД к меткам в потоках можно привязать теги, соответствующие номерам циклов, т.е. в нем потоки являются синхронными. Здесь слово «синхронный» не означает синхронность приема или выдачи данных акторами, а означает лишь то, что модель выполняет алгоритм циклично. Причем поведение модели в каждом цикле одинаково: число использованых и сгенерированных меток равно константе.

Если число меток, использованных каждым входом и сгенерированных каждой вершиной в одном цикле, равно константе, то такой граф называют однородным ГСПД. Синонимом однородного ГСПД является редуцированный граф алгоритма (РГА), название которого происходит от способа его построения из регулярного графа зависимостей по данным [16].

Если число меток, использованных и сгенерированных вершинами, различно, то ГСПД следует назвать неоднородным ГСПД [17]. Неоднородные ГСПД упоминаются в русскоязычной литературе как «многократ-

ные» [18] или «многоскоростные» [19, 20] для систем цифровой фильтрации, т.е. как перевод слова multirate. С помощью ГСПД описываются системы, в которых использовано несколько взаимнократных частот дискретизации.

Неоднородный ГСПД — это компактная модель для задания периодического алгоритма, однако анализировать его сложнее, чем однородный ГСПД. Кроме того, его моделирование усложняется возможностью блокировок. Поэтому для анализа и синтеза ВС на его основе неоднородный ГСПД часто преобразуют в эквивалентный однородный ГСПД [15]. Модель неоднородного ГСПД получила широкое распространение. Например, она является основой известного пакета Matlab-Simulink [21].

ГСПД всегда ассоциировались с алгоритмами ЦОС. Однородный ГСПД взаимно однозначно соответствует сигнальному графу алгоритма ЦОС или вычислительной схеме с периодом вычислений один такт [19, 22, 23].

Разработана такая разновидность ГСПД, в которой вместо одиночных меток рассматриваются группы по N меток, соответствующие массивам отсчетов. Такой ГСПД получил название масштабированного [15]. Для того чтобы задать алгоритм обработки многомерного сигнала, масштабированный ГСПД был обобщен до многомерного. В таком ГСПД метки рассматриваются как многомерные массивы с кратными размерами, а разметка дуг выполнена с помощью векторов, размерность которых совпадает с числом размерностей массивов [15].

Существуют модели, имеющие иерархию циклов выполнения акторов во времени, причем во внутренних циклах число использованных и сгенерированных меток может быть переменным, но при подсчете во внешнем цикле суммарное число меток стабильно. Такие модели получили название циклостатических ГСПД, и в них устранены многие недостатки, присущие моделям представления алгоритмов неоднородными ГСПД [24].

Граф, в котором используются динамические акторы, называется динамическим графом потоков данных (ДГПД). В связи с ограничением на запуск акторов возможности статического ГПД ограничены по сравнению с сетью Петри. По своим особенностям ДГПД эквивалентен машине Тьюринга, которая, в свою очередь, является более универсальной моделью, чем сеть Петри. Вследствие этой универсальности ДГПД достаточно сложно анализировать, так как его поведение в зависимости от потоков данных неопределенно. Поэтому некоторые задачи анализа ДГПД остаются неразрешимыми [15]. Такая модель обработки потоков данных по многим свойствам сповпадает с моделью для реализации языка VHDL, используемой как для моделирования, так и для синтеза ВС [21, 25].

Если ДГПД состоит из акторов, использующих управляющие метки двоичного типа, и регулярных акторов, то такой ДГПД называется бу-

левским ГПД. Более широкую область алгоритмов представляет целочисленный ГПД, акторы которого управляются целыми числами [15]. Акторы этих графов можно описать языком с применением операторов соответственно if-then-else или case. В отличие от ДГПД общего вида поведение булевского или целочисленного ГПД не зависит от способа составления расписания для него, так как эти модели являются разновидностями сети Кана. Булевский ГПД отличается от ГПД Денниса тем, что его потоки представлены не регистрами, а буферами FIFO.

ДГПД может иметь небольшое число динамических вершин, поведение которых можно спрогнозировать, например, определением вероятности их срабатывания. Такие ДГПД позволяют рассчитать статическое расписание для всех вершин, кроме динамических. Эти ДГПД получили название квазистатических [15]. Если возможно вершины подграфа квазистатического ДГПД с динамическими вершинами склеить в вершину, которая будет вести себя, как статическая, то, выполнив такую процедуру, ДГПД можно преобразовать в ГСПД. В этом случае ДГПД называют ДГПД с эффективным поведением [15].

В параметрическом ГСПД акторы могут иметь некоторые наборы функций, а дуги — сответствующие наборы разметок, которые можно динамически изменять независимо от потоков данных на период не меньше одного цикла. Для такого ГСПД можно составить квазистатическое расписание. По свойствам такой граф похож на цикло-статический ГСПД [26]. В моделирующей системе PtolemyII параметрический ГСПД, как и циклостатический целочисленный, внедрен в полном объеме. Эта система эффективна при разработке матобеспечения для ВС обработки мультимедийных данных [21].

Блоковый ГПД является развитием параметрического и масштабированного ГСПД. В нем под меткой понимается пакет или блок данных с определенными структурой и свойствами. Причем акторы и дуги ГПД выполняют свои функции в сответствии с тегами, которые приписаны меткам. Такой ГПД является динамическим и имеет квазистатическое расписание. Блоковый ГПД является удачной моделью для создания алгоритмов обработки цифровых сигналов с переменной структурой, например сжатых видеоданных [27].

Иерархические ГПД. Модели потоков данных классифицируют также по признаку, является ли модель иерархической. В иерархической модели вершина актора сама может быть представлена моделью определенного типа. При использовании языка программирования построение иерархии эквивалентно вызовам соответствующих процедур, выполняющих вычисления моделей нижнего уровня. Если модель нижнего уровня не

вырождена, то она может быть представлена некоторым автоматом с памятью или функцией преобразования. Поэтому часто вычислительные модели классифицируют как модели, имеющие акторы с памятью или без нее.

Можно получить параметрический цикло-статический ГСПД склеиванием множеств вершин однородного ГСПД. При этом новая вершина актора включает в себя несколько вершин исходного графа и поэтому является иерархической. В то же время, новая вершина имеет сложную функцию, которая покрывает функции склеенных вершин исходного графа, и за один цикл последовательно выполняет вычисления этих вершин, т.е. она уже не рассматривается как иерархическая. Такая вершина не содержит памяти, так как потоки с буферами остаются снаружи вершины. Процедура такого преобразования ГСПД получила название сворачивания, а полученный граф — свернутого ГСПД. Такая операция заимствована из оптимизации циклических программ и эквивалентна способу сворачивания циклов. По своей сути свернутый ГСПД изоморфен графу структуры некоторого конвейерного вычислителя, а сворачивание ГСПД используется как метод синтеза таких вычислителей [28].

Для анализа иерархической модели ее желательно преобразовать в одноуровневую модель без иерархии. Существуют модели потоков данных с динамической иерархией, например если вычислительный процесс актора вызывает другие произвольные процессы, как подпрограммы. Хотя такая модель имеет большие креативные возможности для программистов, она является недетерминированной, не дает возможности строить одноуровневую модель и ее анализ затруднителен.

Граф зависимостей по данным (ГЗД) и(или) управлению был введен как естественная модель вычислений, которая удовлетворяет потребности компиляции и распараллеливания программ, написанных на императивных языках [5,29]. В этом графе вершины-операторы связаны дугами, которые соответствуют непосредственным зависимостям по данным между операторами в алгоритме или программе. Если в алгоритме используется принцип однократного присваивания, то дугам соответствуют операнды алгоритма [30]. Во время компиляции программ алгоритм декомпонируется на граф управления и направленные ациклические графы непосредственных зависимостей (известные как DAG), которые представляют собой ГЗД [31]. Выполнение алгоритма, заданного ГЗД, представляет собой последовательный обход вершин в соответствии с их топологической сортировкой [32].

При использовании ярусно-параллельной формы ГЗД разрешается одновременное выполнение операторов, соответствующих одному ярусу графа. Поэтому построение ярусно-параллельной формы ГЗД используется как простейший и понятный способ распараллеливания алгоритмов

[29]. Но ГЗД не приспособлен для задания алгоритмов обработки потоков данных. Такую обработку можно выполнять только при последовательном выполнении алгоритма: как только завершатся вычисления для n-й группы входных данных, можно начать вычисления для $n+1$-й группы.

ГЗД, в котором заданы задержки выполнения операторов или число ступеней конвейерного вычислителя, соответствующего вершине, получил название базовой ВС. Если дуги ГЗД выполняют функцию буфера FIFO, то можно получить уравновешенную базовую ВС, в которой задержка между двумя произвольными вершинами, зависимыми по данным, одинакова для любых маршрутов распространения операндов между ними. Особенность уравновешенной базовой ВС состоит в том, что она соответствует ациклическому алгоритму обработки потоков данных, т.е. эквивалентна однородному ациклическому ГСПД, в котором разметка дуги равна числу ступеней конвейера соответствующей вершины базовой ВС [17, 23].

Во время составления расписания выполнения алгоритма на ГПД строится развертка вычислений, являющаяся ГЗД. Соответственно, если ГЗД — периодический, то он может быть отображен в ГСПД, который можно построить по периоду ГЗД. Таким способом строится РГА, являющийся синонимом однородного ГСПД [16].

ГЗД регулярного алгоритма, например гнезда циклов, представляет собой периодический или решетчатый граф, период которого представляет DAG тела гнезда циклов [17]. Существует методика отображения такого ГЗД в граф систолического процессора [17, 33, 34]. Она является одной из немногих методик, позволивших формальным образом преобразовать вычислительный алгоритм в параллельную структуру специализированного вычислителя. Следует заметить, что граф систолического процессора, по сути, является однородным ГСПД и, следовательно, ГСПД можно строить, используя эффективные и формальные методы синтеза систолических процессоров [16].

Проверка на отсутствие блокировок. Для того чтобы заранее определить, будет ли составление расписания успешным, необходимо проверить модель на отсутствие блокировок. Такую проверку можно выполнить аналитически, если акторы представляют функции, отображающие входные потоки данных в выходные. Блокировка модели наступает тогда, когда вследствие блокировки отдельных акторов с некоторого момента модель не в состоянии повторять период алгоритма и генерацию сигналов произвольной длины.

Наиболее конструктивным и общим является метод проверки, основанный на теории решеток. Цепочки (сигналы) C возможных потоков

данных a_i в ГПД формируют частичный порядок, который для многих ГПД является завершенным. Согласно этому методу составляют уравнение $a_i = F(a_{i-1})$, которое связывает поток данных a_i на i-й итерации с потоком данных a_{i-1} на i–1-й итерации на основе композиции функций всех акторов, входящих в замкнутые циклы ГПД. Если функция F — монотонна, то модель ГПД не имеет блокировок и вычисления в модели могут выполняться сколь угодно долго. Это эквивалентно тому, что можно найти решение уравнения отображения $F(a) = a$, т. е. крайнюю неподвижную точку отображения, которая равна единственному крайнему верхнему пределу цепочки C.

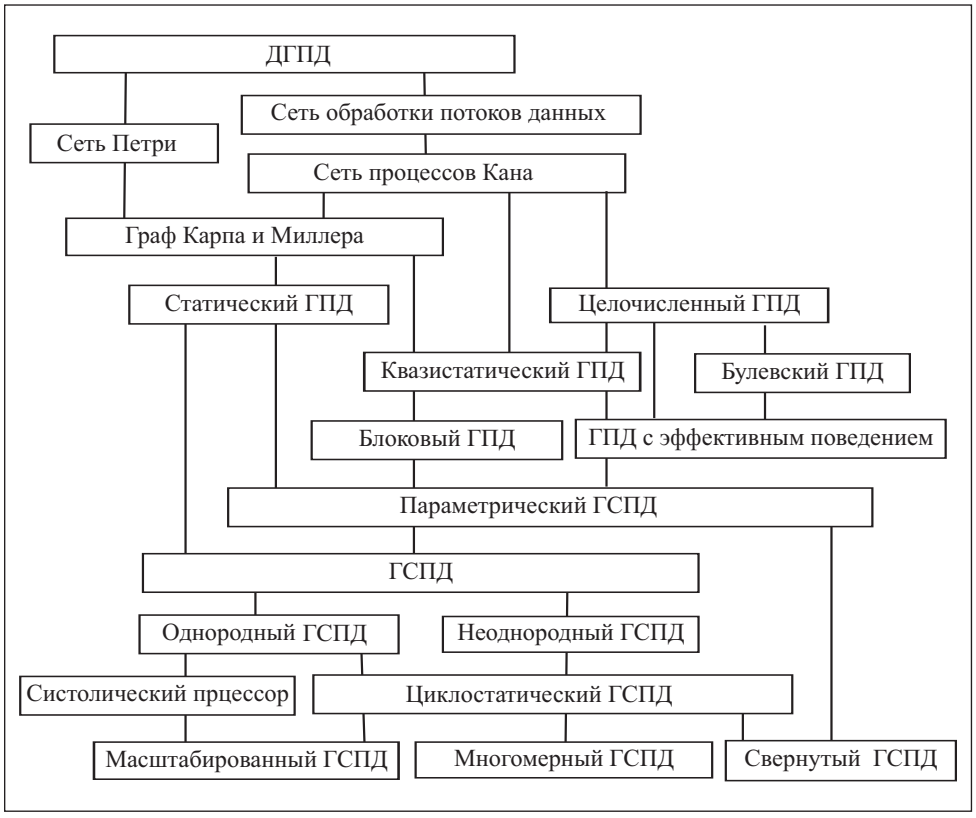
Суть метода можно раскрыть на простом примере итеративного процесса вычисления аппроксимации функции $f(x)$ рядом Тейлора, в котором в каждой следующей итерации получается уточненный результат. Тогда a_i — это приближение на i-й iтерации, $C = (a_0, a_1, a_2 ...)$, а $f(a)$ — крайний верхний предел, равный точному значению функции $f(x)$. Этот метод можно использовать для многих моделей, включая сеть обработки потоков данных и сеть Кана [21].

Для ГПД Карпа и Миллера отсутствие блокировок эквивалентно наличию целочисленного решения задачи линейного программирования с неравенствами ограничения числа меток в дугах [13].

Для модели неоднородного ГСПД отсутствие блокировок определяют при решении системы уравнений баланса $\Gamma\mathbf{r} = 0$. Здесь Г — топологическая матрица, имеющая свойства матрицы инцидентности графа и учитывающая число использованных и выдаваемых меток каждой вершиной; **r** — вектор повторений, указывающий, сколько раз каждая вершина должна срабатывать при выполнении одного цикла алгоритма. Однако в этой модели, как и в ГПД Карпа и Миллера, блокировка возможна, если число меток в дугах модели в ее начальном состоянии недостаточно. Такая же проверка возможна для булевского ГСПД, если в матрицу Г вставлены вероятности выдачи и использования меток динамическими акторами.

Однородный ГСПД никогда не блокируется при наличии меток начального состояния в буферах, входящих в циклы графа, и поэтому блокировки в нем можно не проверять.

Таким образом, аналитическая проверка потоковой модели на отсутствие блокировок несложна для ГСПД и усложняется с переходом к более обобщенным моделям ГПД. Так, для сети потоков данных доказательство отсутствия блокировок сводится к процессу поиска неподвижной точки отображения пространства сигналов, который является весьма сложным и связан с необходимостью учитывать все возможные переключения динамических акторов [21]. Наиболее часто используемый способ проверки блокировок — динамическое составление расписания и моделирование сети.



Выбор алгоритма обработки потоков данных. Основные свойства моделей обработки потоков данных приведены в таблице. Взаимоотношение графовых моделей обработки потоков данных представлено на рисунке в виде графа. Из приведенных данных можно сделать следующие выводы.

Граф потоков данных, используемый для задания различных алгоритмов обработки потоков данных, имеет много разновидностей, созданных для упрощения его анализа, адаптации к отдельным классам алгоритмов, большей выразительности их представления. Алгоритм, заданный ГПД, показывает, как получается результирующий поток данных и не определяет точный порядок вычислений. Отображение такого алгоритма в ВС состоит в нахождении оптимального расписания и назначении вершин графа в процессорные элементы ВС. Поиск оптимального отображения является *NP*-полной задачей. Выбор разновидности ГПД зависит от предметной области применения алгоритма (моделирование или обработка данных (сигналов), характер сигналов) и способа его исполнения (аппаратного или программного).

Для использования в системном синтезе аппаратных устройств обработки потоков данных наиболее приспособлена модель ГСПД и ее усовершенствования — параметрический, цикло-статический и многомерный ГСПД. Могут быть также использованы модели квазистатического ДГПД и ДГПД с эффективным поведением, для которых можно составить расписание со свойствами статического расписания. Иерархические модели, состоящие из упомянутых графов, также могут быть использованы, поскольку они допускают развертывание до одноуровневой модели.

Следует больше внимания уделять однородному ГСПД, для которого характерны простой анализ, возможность эквивалентного преобразования в него других типов ГСПД. Известный подход отображения регулярных ГЗД, в том числе ГЗД систолических алгоритмов, можно заменить отображением ГСПД, который является эффективной формой представления таких ГЗД. Динамические графы потоков данных, как и сети Петри, могут эффективно внедряться только в ВС с программной реализацией алгоритмов, так как для них требуется динамическое составление расписания, которое может оказаться безуспешным вследствие возможных блокировок.

Применение ГСПД для синтеза конвейерных ВС. Конвейерная ВС представляет собой многоступенчатый аппаратный вычислитель, обраба-

Модель	Расписание	Синхронность модели	Функция запуска актора	Аналитическая проверка блокировок
Сеть обработки потоков данных	Динамическое	Асинхронная	Набор правил	Поиск неподвижной точки отображения
Сеть Кана	"	"	Блокировка чтения	То же
Граф Карпа и Миллера	"	"	$W_k \geq T_k$	Решение задачи линейного программирования
Целочисленный ГПД	"	"	$W_k = T_k$ для статических акторов, case — для динамических	Решение $\Gamma \mathbf{r} = 0$
Квазистатический ГПД	Квазистатическое	"	То же	То же
Неоднородный ГСПД	Статическое	Акторы — асинхронные, потоки — синхронные	$W_k = T_k$	"
Однородный ГСПД	"	То же	То же	Не требуется
Систолический процессор	"	Синхронная	По синхросигналу	Не требуется

тывающий потоки данных. Его структура может быть получена отображением ГСПД. Выше было упомянуто, как конвейерная ВС получается единичным отображением свернутого ГСПД. В [35] предложено представлять однородный ГСПД в многомерном пространстве. Это дало возможность целенаправленно искать как структуру ВС, так и расписание выполнения алгоритма в ней. В [36, 37] этот подход усовершенствован с целью минимизации аппаратных затрат ПЛИС, в которой реализуется конвейерная ВС.

Динамический граф потоков данных с эффективным поведением отображается в конвейерную ВС методом, предложенным в [37]. Этот метод дает возможность формально отображать алгоритмы с операторами управления в структуру конвейерных ВС с заданным периодом вычислений, имеющих минимизированные аппаратные затраты и высокую тактовую частоту. Поскольку результатом отображения является описание ВС на языке VHDL, метод дает возможность не строить собственно структуру ВС и расписание выполнения алгоритма, а переадресовать это задание компилятору-синтезатору проекта для ПЛИС.

Выводы. Граф потоков данных представляет собой естественную модель для задания алгоритмов обработки таких потоков. Динамические ГПД имеют широкие возможности для задания алгоритмов, но их анализ усложнен и в них могут происходить блокировки. Поэтому они используются в основном в системах с динамическим расписанием и не приспособлены для синтеза конвейерных ВС.

Конвейерные ВС следует проектировать отображением ГСПД или ГПД с эффективным поведением и квазистатических ГПД, имеющих ряд свойств, таких же, как у ГСПД. Меньшая выразительность и большая трудоемкость представления алгоритма на модели однородного ГСПД компенсируется тем, что при представлении такого ГСПД в многомерном пространстве его отображение в конвейерную структуру выполняется формально с получением минимизированных аппаратных затрат.

Properties of various computational models and recommendations are proposed for the selection of such a model which is fitted for the pipeline computer designing.

1. *Минский М.* Вычисления и автоматы. — М. : Мир. — 1971. — 264 с.
2. *Berry G.* Real Time Programming: Special Purpose or General Purpose Languages // Information Processing / G. Ritter, Ed. Elsevier Science Publishers B. V. (North Holland) — 1989. — **89.** — P. 11—17.
3. *Lee E. A.* Embedded Software //Advances in Computers / M. Zelkowitz. Ed. — London : Academic Press, 2002. — **56.** — 29 p.

4. *Сергиенко А. М.* 5,5 десятилетий цифровой обработки сигналов // Argc&Argv. —2006. — № 1. — С. 19—25.
5. *Lee E. A.* The Problem with Threads // IEEE Computer. — 2006. — **39**, № 5. — P. 33—42.
6. *Котов В. Е.* Введение в теорию схем программ. — Новосибирск : Наука, 1978. — 258 с.
7. *Петерсон Дж.* Теория сетей Петри и моделирование систем. — М. : Мир,1984. — 264 с.
8. *Котов В.Е.* Сети Петри. — М. : Наука, 1984. — 160 с.
9. *Maciel P., Barros E., Rosenstiel W.* A Petri Net Model for Hardware/Software Codesign // Design Automation for Embedded Systems. — 1999. — № 4. — P. 243—310.
10. *Eles p., Kuchinski K., Peng Z.* System Synthesis with VHDL. — Kluwer Academic Pub. — 1998. — 370 p.
11. *Ачасова С. М., Бандман О. Л.* Корректность параллельных вычислительных процессов / Отв. ред. Н.Н. Миренков. — Новосибирск : Наука, Сиб. отд-ние, ВЦ. — 1990. — 252 с.
12. *Lin B.* Efficient Compilation of Process-based Concurrent Programs without Run-time Scheduling // Proc. of Design, Automation, and Test in Europe (DATE). — Paris. — 1998. — P. 211 — 217.
13. *Вальковский В. А., Котов В. Е., Миклошко Й. и др.* Алгоритмы, математическое обеспечение и архитектура многопроцессорных вычислительных систем. — М. : Наука, — 1982. — 340 с.
14. *Системы* параллельной обработки /под ред. Д. Ивенса, — М. : Мир. — 1985. — 413 с.
15. *Edwards S., Lavagno L., Lee E. A., Sangiovanny-Vincentelli A.* Design of Embedded Systems: Formal Models, Validation and Synthesis //Proc. of the IEEE. —1997. — **85**, № 3. — P. 366—390.
16. *Рао С. К., Кайлат Т.* Регулярные итеративные алгоритмы и их реализация в процессорных матрицах // ТИИЭР. — 1988. — **76**, № 3. — С. 58—69.
17. *Воеводин В. В.* Математические модели и методы в параллельных процессах. — М : Наука, 1986. — 296 с.
18. *Гольденберг Л. М. и др.* Цифровая обработка сигналов: Справочник. — М. : Радио и связь, 1985. — 312 с.
19. *Сергиенко А. Б.* Цифровая обработка сигналов (2-е изд.). — СПб. : Питер, 2006. — 751 с.
20. *Вайдьнатхан П. П.* Цифровые фильтры, блоки фильтров и полифазные цепи с многочастотной дискретизацией. Методический обзор // ТИИЭР, — 1990. — **78**, № 3. — С. 77—120.
21. *Lee E. A., Neuendorffer S.* Concurrent Models of Computation for Embedded Software //IEE Proc. Comput. Digit. Tech. — 2005. — **152**, № 2. — P. 239—250.
22. *Оппенгейм А. В. Шафер Р. В.* Цифровая обработка сигналов. — М. : Связь, 1979. — 416 с.
23. *Сергиенко А. М.* VHDL для проектирования вычислительных устройств. — Киев : Диасофт, 2003. — 200 с.
24. *Bilsen G., Engels M., Lauwereins R., Peperstraete J.* Cyclo-static Dataflow //IEEE Trans. on Signal Processing. — 1996. — **44**, № 2. — P. 397—408.
25. *Сергиенко А. М.* Особенности VHDL как языка параллельного программирования // Электрон. моделирование. — 2003. — **25**, № 3. — С. 115—123.
26. *Bhattacharya B., Bhattacharyya S. S.* Parameterized Dataflow Modeling for DSP Systems// IEEE Trans. on Signal Processing. — 2001. — **49**, № 10. — P. 2408—2421.
27. *Ko D., Bhattacharyya S. S.* Modeling of Block-based DSP Systems. // Proc. of the IEEE Workshop on Signal Proc. Systems. Seoul, Korea. —2003. —August. —P. 381—386.
28. *Parhi K. K., Wang C. Y., Brown A. P.* Synthesis of Control Circuits in Folded Pipelined DSP Architectures // IEEE J. Solid-St. Circ. — 1992. — **27**. — P. 29 —43.

29. *Поспелов Д. А.* Введение в теорию вычислительных систем. — М. : Сов.радио, 1972. — 280 с.
30. *Сир Ж. -К.* Метод потока операндов в моногопроцессорных системах типа MIMD // Системы параллельной обработки / Под ред. Д.Ивенса. — М. : Мир, 1985. — 413 с.
31. *Ахо А., Сети Р., Ульман Дж.* Компиляторы: принципы, технологии, инструменты. — М.: Изд. дом «Вильямс», 2003. — 768с.
32. *Евстигнеев В. А.* Применение теории графов в программировании. — М. : Наука, 1985. — 350 с.
33. *Кун С.* Матричные процессоры на СБИС. — М. : Мир, 1991. — 672 с.
34. *Каневский Ю. С.* Систолические процессоры. — Киев : Техніка, 1991. — 172 с.
35. *Каневский Ю. С., Овраменко С. Г., Сергиенко А. М.* Отображение регулярных алгоритмов в структуры специализированных процессоров // Электрон. моделирование. — 2002. — **24**, № 2. — С. 46—59.
36. *Симоненко В.П., Сергиенко А.М.* Отображение периодических алгоритмов в программируемые логические интегральные схемы // Там же. — 2007. — **29**, № 2. — С. 49—61.
37. *Сергиенко А.М.* Синтез структур для выполнения периодических алгоритмов с операторами управления // Вістн. Національного техничного університету України «КПІ». Сер. Інформатика, управління і обчислювальна техніка. — 2008. — № 47. — С. 62— 68.

Поступила 15.07.08

СИМОНЕНКО Валерий Павлович, д-р техн. наук, проф. Национального технического университета Украины «КПИ», который окончил в 1965 г. Область научных исследований — организация вычислительных процессов в вычислительных системах.

СЕРГИЕНКО Анатолий Михайлович, канд. техн. наук, ст. науч. сотр. Национального технического университета Украины «КПИ», который окончил в 1981 г. Область научных исследований — отображение алгоритмов в структуры вычислительных средств, цифровая обработка сигналов.