ПРОБЛЕМЫ РЕАЛИЗАЦИИ МАССОВОГО ДИНАМИЧЕСКОГО ПАРАЛЛЕЛИЗМА. II

А.В. Махиборода, А.В. Ильичёв, А.А. Подобин, А.В. Царёв

Департамент прикладной математики МИЭМ,
Национальный исследовательский университет «Высшая школа экономики»

makhiboroda@yandex.ru

Поступила 17.05.2016

В первой части статьи было отмечено, что освоение массового динамического параллелизма становится основным инструментом обеспечения устойчивого роста производительности вычислительных средств. Показано также, что на пути реализации массового параллелизма имеются два основных препятствия. Это специфические затраты на поддержку параллелизма, которые растут опережающими темпами с ростом числа процессорных элементов, и низкая производительность обменной среды, поддерживающей межпроцессорные пересылки данных.

Во второй части проводится оценка прогнозов развития обменной среды в части совершенствования её алгоритмов и повышения пропускной способности. Делаются экспертные заключения о возможностях расширения диапазона эффективного параллелизма до значений в пределах сотен процессоров. При этом показатель в несколько тысяч процессоров на кристалле уже достигнут, а данные о текущем состоянии технологии позволяют прогнозировать рост числа процессоров до десятков тысяч и более на ближайшие несколько лет. Поиск путей дальнейшего расширения диапазона эффективной реализации параллелизма является актуальным и направлен на освоение принципиально новых форм организации вычислений. Излагаются основные идеи архитектуры самоопределяемых данных, и даётся краткий очерк построения математического аппарата дискретной динамики. Понятийный аппарат дискретной динамики позволяет сформулировать и решить задачу программирования процессов в архитектуре самоопределяемых данных.

УДК 681.3.06

Введение

В предыдущем разделе мы рассмотрели конкретный пример, который позволяет определить некую общую тенденцию и дать качественную оценку динамики роста производительности параллельных систем при наращивании параллелизма в широком

диапазоне значений. При организации работы параллельных структур кроме основного объёма вычислительных работ, распределённых на множестве обрабатывающих процессоров, неизбежно возникают дополнительные издержки в виде затрат на обслуживание параллелизма. Издержки обслуживания параллелизма в основном представлены пересылками данных между процессорными элементами и процедурами доступа к общей памяти и всегда растут с ростом параллелизма. Следовательно, динамика роста производительности параллельных систем формируется как результат наложения двух взаимоисключающих тенденций – сокращения времени выполнения задачи вследствие параллелизма и роста издержек на обслуживание параллелизма. Длительный монотонный рост производительности при наращивании числа каналов обработки в данной ситуации невозможен. В общем случае динамика изменения производительности параллельной системы будет проходить стадию роста, затем входить в насыщение и далее падать. Реальные задачи проектирования и эксплуатации параллельных систем заключаются в том, чтобы вывести точку насыщения графика роста производительности в область более высоких значений и более полного использования параллелизма. К поставленной цели можно двигаться по двум направлениям – увеличивать производительность обменной среды и снижать объёмы обменных операций. Увеличение продуктивности межпроцессорных обменов задача инженерная, это выбор физической среды, повышение полосы пропускания, совершенствование алгоритмов и протоколов обмена. Сокращение объёмов обменных операций это более сложная задача, которая носит системный характер и требует анализа и корректировки архитектурных принципов и форм организации вычислительных процессов.

Цель настоящей статьи состоит в том, чтобы обозначить границы роста эффективности параллелизма вследствие совершенствования обменной среды, а также обосновать необходимость и наметить перспективы разработки принципиально новой архитектурной идеи, поддерживающей массовый динамический параллелизм.

1. Основные характеристики и направления развития коммутационных сетей

Начиная с определённого момента в процессе роста числа транзисторов на кристалле, разработчики чипов стали осваивать новое направление развития – создание многоядерных изделий, в которых число процессорных элементов исчисляется сотнями и тысячами. На период с 2020 года и далее прогнозируется достижение показателя степени интеграции в триллион транзисторов на кристалле, что позволит довести число ядер до миллиона [1]. В проектировании и производстве процессорных элементов накоплен огромный опыт, обеспечивающий большое разнообразие типов архитектур, ориентированных на разные виды применений. В тоже время средства обеспечения межпроцессорных обменов не получили достаточного развития и на данный момент являются самым консервативным и проблемным сегментом внутренней структуры кристалла. Очевидно, что при организации согласованной работы больших ансамблей процессорных элементов ведущая роль отводится средствам поддержки межпроцессорных обменов и коммутации. Наиболее полно тематика построения высокоскоростных коммутационных сетей межпроцессорных обменов представлена в области производства и эксплуатации суперкомпьютеров.

Рассмотрим основные характеристики и направления развития средств коммутации и межпроцессорных обменов, которые в дальнейшем будем называть сети интерконнекта. Для нас важно оценить две принципиально отличающиеся технологии построения обменной среды – с коммутацией каналов и коммутацией пакетов. Традиционной считается технология коммутации каналов. При коммутации каналов имеется некий пул процессорных элементов и над ним строится обменная среда, которая должна обеспечить возможность соединения каждого элемента с каждым для осуществления обмена данными. При значительных объёмах пула обеспечить полнодоступное соединение всех участников обмена со всеми технически невозможно. Поэтому строятся разнообразные системы переключателей с частичным доступом, в которых пара обменивающихся абонентов монополизирует коммутационные ресурсы на время сеанса обмена данными. При этом часть путей обмена на время сеанса становится недоступной для других участников пула, а часть может одновременно обслуживать другие запросы на обмен. Технология коммутации каналов предполагает, что в составе системы имеются средства управления коммутацией и есть протокол, определяющий алгоритмику обменов. Участники пула должны посылать в систему управления запросы на обмен, система управления должна прокладывать путь обмена и после завершения сеанса восстанавливать исходное состояние коммутационной среды. Технические характеристики коммутационных систем определяют объём обслуживаемого пула абонентов, полосу пропускания и скорости пересылки данных, временные затраты на выполнение протокольных событий доступа к сети, вероятность отказа в доступе и среднее время ожидания в очереди на обслуживание. Важное обстоятельство заключается в том, что динамика функционирования обменной среды носит стохастический характер. Это может создавать определённые проблемы при построении параллельных вычислительных систем. В ряде приложений необходимо учитывать, что синхронные алгоритмы погружаются в аппаратную среду со стохастической динамикой, и это может привести к нежелательным последствиям.

Сети с технологией коммутации пакетов состоят из множества узлов, каждый из которых имеет фиксированное число постоянных некоммутируемых связей со смежными узлами. Данные оформляются как сетевые пакеты, которые могут продвигаться по сети от узла к узлу различными маршрутами. В узлах сети располагается оборудование, управляющее продвижением пакетов. В узлах сети могут располагаться, в том числе и процессорные элементы. Отличительный признак сетей с коммутацией пакетов состоит в том, что обменная среда может совмещаться со средой обработки. Такие сети легко масштабируются и хорошо приспособлены для построения больших многопроцессорных систем. Топология пакетной сети обеспечивает связность всех элементов процессорного пула, поскольку для каждой пары обменивающихся процессорных элементов всегда есть, связывающий их путь.

Важнейшей характеристикой коммутационной сети является топология её построения. Имеется огромный опыт разработки разнообразных топологий коммутационных сетей и обширная литература по этой теме, достаточно ознакомиться, например, с монографией [2]. Мы прокомментируем только некоторые из них, которые наиболее популярны в настоящее время у разработчиков суперкомпьютеров и мультипроцессорных систем. Сравнительные характеристики и анализ различных систем коммутации, применяемых в суперкомпьютерах можно найти в [3, 4]. На рис. 1 приведена структура коммутационной сети, известная под названием «толстое дерево».

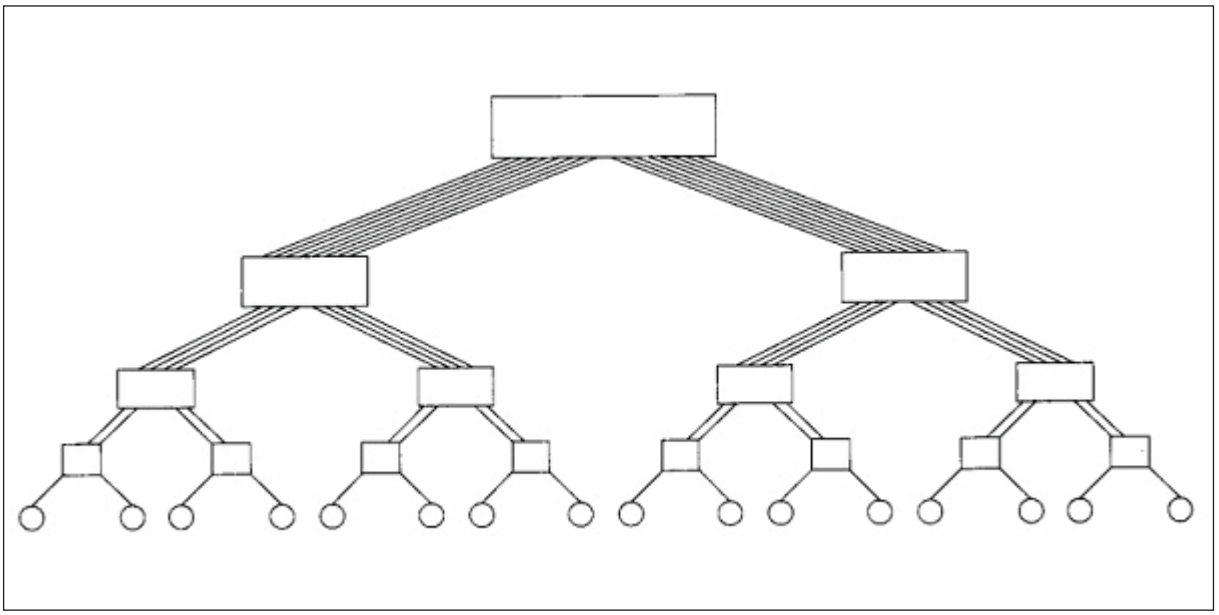


Рис. 1. Схема объединения узлов коммутаторами в топологии «толстое дерево».

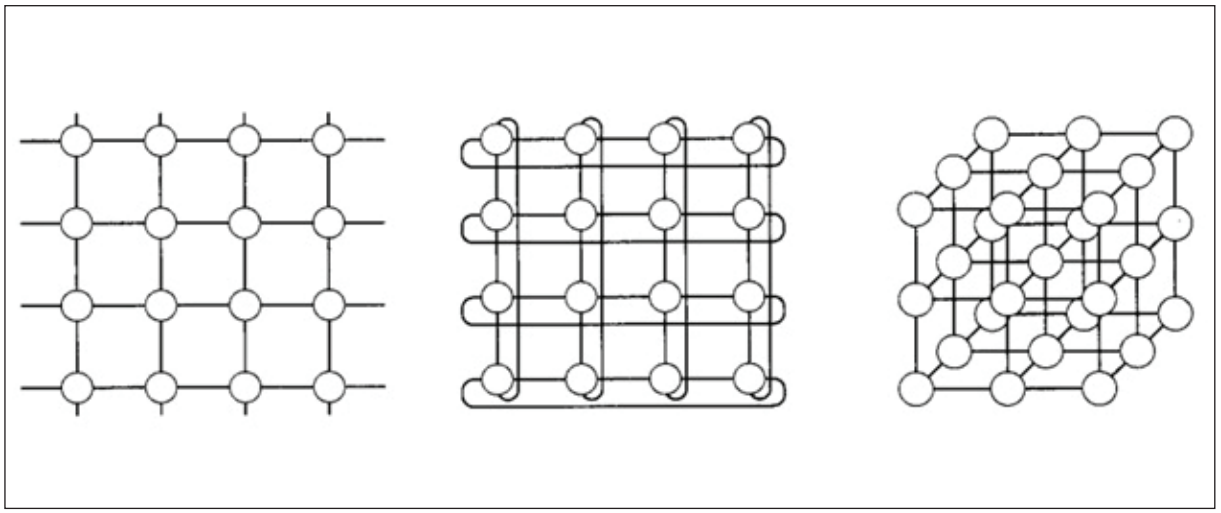


Рис. 2. Схема объединения узлов непосредственно друг с другом в топологии решётка и тор.

Это типичный пример построения сети по технологии коммутации каналов. Сеть «толстое дерево» имеет ограничения на рост объёма процессорного пула. «Толстое дерево» эффективно работает при объединении сотен процессорных элементов, при переходе тысячного рубежа издержки начинают расти взрывообразно. Более перспективной для построения мультикомпьютерных систем оказываются топологии решётки, тора и многомерных торов, изображённые на рис. 2.

Активно разрабатываются гибридные структуры, сочетающие разные топологии в иерархических построениях, это структуры «стрекоза» и «бабочка» [5]. Проектирование коммутационных сетей решает сложные и противоречивые задачи – рост объёмов процессорного пула вызывает рост размерности и сложности сети, а это приводит к увели-

чению числа переключателей, удлинению маршрутов движения данных и к усложнению алгоритмов маршрутизации. Складывается парадоксальная ситуация, при которой сталкиваются две взаимоисключающие тенденции. С ростом размерности сети растёт число абонентов и путей передачи данных, что означает увеличение пропускной способности сети в целом. В тоже время увеличивается сложность маршрутов для каждого конкретного сеанса обмена. Рост числа переключателей и буферных зон на пути движения данных и усложнение алгоритмов маршрутизации существенно понижают скорость обменов, что не способствует росту суммарной продуктивности сети. Ситуацию пытаются переломить применением дорогостоящих высокочастотных элементов и оптических кабелей. При этом приходится искать компромиссы, связанные с ростом энергопотребления и высокой стоимостью аппаратуры [6, 7, 8].

По имеющимся на данный момент сведениям во внутренней среде кристалла для построения многоядерных структур используется каскадная многоуровневая коммутация на базе общей шины. В перспективе принятие решений по структуре внутренней обменной среды будет обусловлено выбором архитектуры кристалла и новыми возможностями технологии многослойной 3D компоновки. Можно предполагать, что предпочтительными окажутся топологии решётки и тора. Для понимания перспектив становления многоядерных структур на кристалле необходимо сделать оценки эффективности использования параллелизма в широком диапазоне значений числа процессорных элементов. С этой целью был сделан пробный расчёт ожидаемых значений коэффициента ускорения от параллелизма для конкретного алгоритма быстрого преобразования Фурье (БПФ) с прореживанием по времени на 2^{16} = 65536 точек. Разметка шкалы N (числа процессорных элементов) была сделана по некоторым правдоподобным предположениям наращивания N по степеням двойки с определённым вариантом многоуровневой компоновки на иерархии общих шин. Так, например, для первой точки шкалы компоновка представляет кластер из 16 процессоров на общей шине, для второй точки компоновка достигается объединением 16 кластеров на шине второго уровня и т. д. Исходные данные расчёта приведены в таблице 1.

Таблица 1

Количество ядер	Время выполнения	Коэффициент ускорения
1	10485760	1
16	917824	11,42
16x16 (256)	446032	23,51
16x32 (512)	363088	28,88
32x32 (1024)	672640	15,59
16x16x8 (2048)	672770	15,59
16x16x16 (4096)	601024	17,45
16x16x32 (8192)	600480	17,46
16x32x32 (16384)	605750	17,2
32x32x32 (32768)	702640	14,92

Время выполнения алгоритма рассчитывалось так же как и в первой части настоящей статьи, где суммировалось время выполнения счета, время выполнения процедур доступа к сети и время передачи данных. Для каждой точки расчет был индивидуальным, поскольку затраты времени на обмен данными определялись структурой шинной компоновки (см. первый столбец таблицы). Относительно характеристик шины были сделаны идеальные предположения о том, что шина всегда доступна и задержек доступа не бывает, шина широкополосная и ведёт обмен машинными словами, тактовая частота шины равна тактовой частоте процессора, а затраты на протокол доступа минимальны и равны одному такту. Полученный график роста коэффициента ускорения приведен на рис. 3.

Пик производительности достигается при числе процессоров равном 512 и позволяет получить ускорение в 28.88 раз. Далее начинается неуклонное падение производительности. Провал графика при значениях N 1024 и 2048 объясняется структурой компоновки и условиями балансирования коротких и длинных передач. Но по мере продвижения в область больших значений N доминантой становится время пересылки данных, а все сопутствующие компоненты и эффекты становятся относительно малыми. Возрастающий параллелизм производит возрастающие объёмы межпроцессорных обменов. При этом самые высокоскоростные и широкополосные сети интерконнекта не в состоянии изменить ход графика ускорения существенным образом. Напоминаем, что приведенный расчет представляет частный случай оценки условий распараллеливания конкретного алгоритма, но он позволяет выявить общую тенденцию. А суть этой общей тенденции такова, что системы с массовым параллелизмом на уровне десятков и сотен

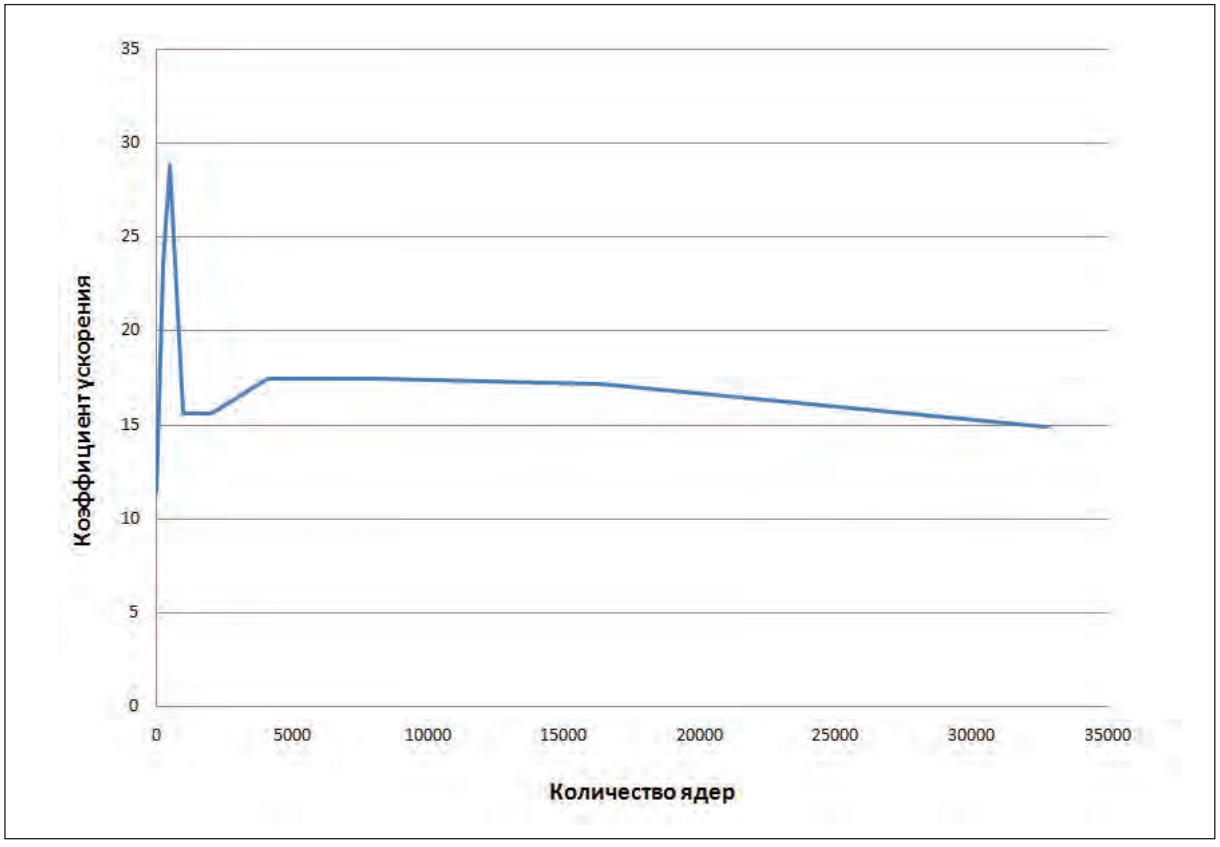


Рис. 3. График роста коэффициента ускорения для выполнения алгоритма БПФ

тысяч процессорных ядер будут очень чувствительны к типам решаемых задач и не смогут исполнить роль универсальных вычислительных средств широкого назначения.

2. Оценки эффективности организации вычислений

Расположение точки насыщения роста производительности определяется соотношением временных затрат на полезную работу и на издержки разных видов. Естественным образом возникает вопрос – почему в рассмотренных нами в первой части статьи примерах динамика роста производительности параллельных систем ограничивается столь низкими значениями коэффициентов ускорения и степени параллелизма. Это 3-х кратное ускорение при 6 процессорных элементах в примере с моделированием редукционно-потоковой архитектуры и 6-х кратное ускорение при 12 узлах обработки во втором примере [9].

Если определяющим фактором являются соотношения временных затрат на полезную работу и на издержки необходимо провести оценку существующих форм организации вычислений с целью определения источников издержек и возможности их радикального сокращения. Необходимо учитывать, что при распараллеливании процессов издержки, свойственные одному процессорному элементу размножаются кратно показателю параллелизма. Кроме того, порождаются дополнительные издержки обслуживания параллелизма, которые прибавляются к общему грузу издержек и таким образом суммарные издержки растут опережающим темпом. В современных условиях, когда произошло введение критерия эффективности кристалла по показателю числа операций на единицу затрат энергии учет издержек и определение их соотношения к полезной работе кристалла становится критическим показателем, от которого зависит возможность реализации массового параллелизма.

Рассмотрим основные положения классической архитектуры с точки зрения оценки издержек и эффективности организации процессов. Напомним, что все промышленно выпускаемые в настоящее время вычислительные средства построены по классической архитектуре. Неклассические архитектуры существуют как проекты или модели и не переходят в стадию промышленной реализации. Сущность архитектурной идеи наиболее полно воплощается в знаковой системе, средствами которой осуществляется программирование машины и управление внутренними процессами.

В архитектуре классической машины фон Неймана управление вычислительными процессами осуществляет простейшая знаковая система, называемая линейным императивным языком. Элементами императивного языка являются команды, которые в совокупности образуют функционально полный набор базисных функций. Функциональная полнота базисного набора команд позволяет декларировать тезис алгоритмической универсальности машины - средствами системы команд можно запрограммировать любой алгоритм. Программа представляет собой линейную последовательность команд.

Основой аппаратной реализации знаковой системы является машинное слово – это битовая строка определённой длинны. Длина или разрядность машинного слова это одна из основных характеристик машины. В языке машины существует две версии интерпретации машинных слов – слова команды и слова операнды. Команда реализована на аппаратном уровне как битовая строка, разбитая на несколько битовых секций или полей, имеющих своё содержательное назначение. Далее для экспертной оценки эффективности организации процессов мы будем использовать некую гипотетическую ма-

шину с фиксированным трёхадресным форматом команды и с памятью, в которой осуществляется адресный доступ к машинным словам. Это не соответствует реальной организации современных машин. Так, например доступ в память осуществляется по байтам, что необходимо для поддержки переменных форматов команд и данных. Команды и данные размещаются в разных блоках памяти и читаются независимо по разным каналам доступа. И таких отличий от предлагаемой к рассмотрению гипотетической машины огромное число, но все эти отличия не меняют суть архитектурной идеи, а их учёт не только осложнит, но и сделает невозможной экспертную оценку эффективности организации вычислений.

Первое битовое поле кодирует операцию и называется КОП. КОП интерпретируется как число в двоичной системе и далее дешифрируется аппаратурой по таблице соответствия. Следующие три поля это так называемые адресные поля, которые обозначаются как A_1, A_2 и A_3. Это числа в двоичной системе счисления, по которым осуществляется доступ в память. Память состоит из массива регистров, имеющих индивидуальные номера или адреса. В регистрах памяти размещаются машинные слова. По предъявлению адреса осуществляется доступ к заданному регистру для чтения либо записи. Содержательно команда представляет элементарную процедуру вычислительного процесса, поддерживающую бинарную операцию с двумя входными операндами и одним результатом. Так, например, A_1 и A_2 могут обозначать адреса, по которым в памяти размещаются два входных операнда, а A_3 обозначает адрес для записи результата. Процессор связан с памятью каналом доступа, память служит местом размещения машинных слов. Память не различает слова как команды или операнды и это позволяет записывать в неё и программы и данные. Программист условно разделяет пространство памяти на сегменты для раздельного размещения программ и данных. Вычислительный процесс представляет собой поток обменов процессора с памятью, в котором циркулируют машинные слова, имеющие статус команд и операндов. Различение машинных слов на команды и операнды происходит при чтении. При этом команды читаются в устройство управления, а операнды в арифметико-логический блок.

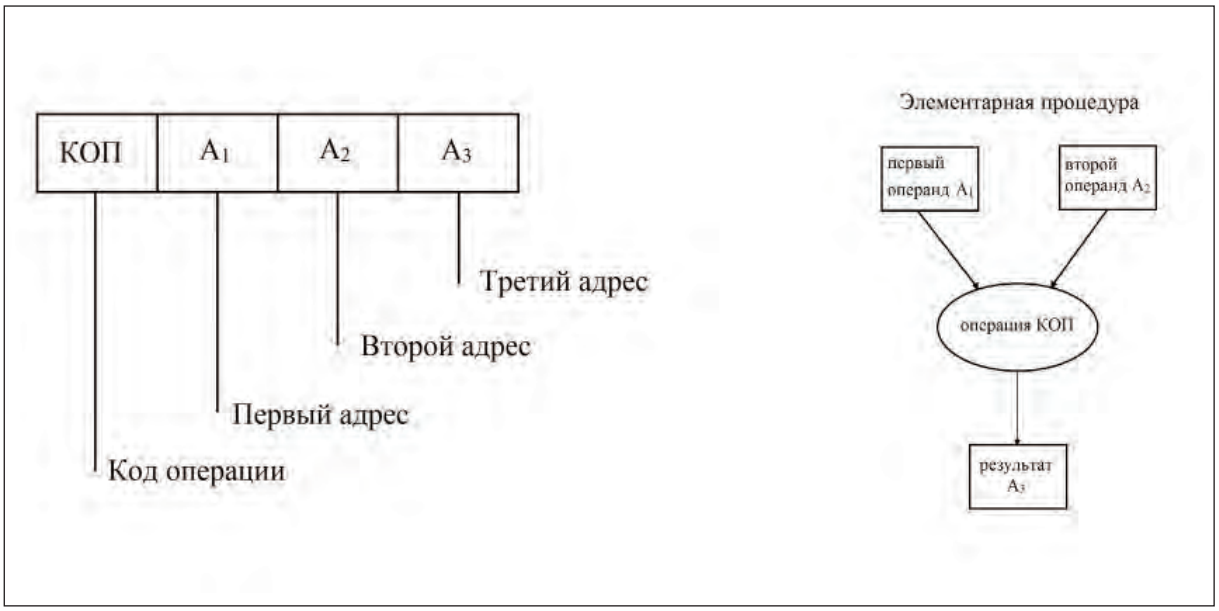


Рис. 4. Структура команды

Изложенное до сих пор позволяет зафиксировать одно очень важное следствие – знаковая система классической машины не является языком описания процессов вычисления и обработки данных. Язык машины является средством описания процессов преобразования состояний памяти. Команды, из которых состоит программа, привязаны не к данным, а к адресам памяти. Для того что бы процесс преобразования состояний памяти превратился в процесс обработки данных необходимо принять меры, обеспечивающие размещение нужных данных в нужных адресах памяти строго к тому моменту, когда соответствующая команда будет активирована и осуществит обращения к заданным адресам. Эти меры осуществляются как на аппаратном, так и на программном уровнях и являются основными источниками издержек и накладных потерь, сопровождающих вычислительный процесс. Для оценки эффективности организации процессов в гипотетической машине нам необходимо рассмотреть временную диаграмму и установить соотношение временных затрат на полезную работу и на сопутствующие обеспечивающие события, которые являются накладными потерями ресурса времени. В итоге нам необходимо оценить коэффициент полезного действия (КПД) машины как процентное соотношение разных работ на временной шкале.

Основным элементом временной диаграммы машины является такт выполнения команды, который состоит из следующих пяти микротактов: чтения команды, чтения первого операнда, чтения второго операнда, выполнения операции и записи результата. Сделаем допустимое в данном случае огрубление и будем считать, что все микротакты имеют равное время выполнения. Полезным квантом временной диаграммы считается только один микротакт – выполнение команды. Все остальные являются обеспечивающими и составляют накладные потери или издержки. Тогда получается, что на уровне выполнения одной команды языка машины во временной диаграмме полезный квант занимает 20%, а на издержки приходится 80%.

При построении простейшей процедуры некий стереотипный фрагмент программы повторяется многократно. Например, обрабатывается таблица, имеющая некоторое число атрибутов в строке и несколько сотен строк. Процедура состоит из программы обработки строки и цикла, повторяющего эту программу многократно по числу строк. Мы знаем, что адресные поля команды привязывают её к конкретным регистрам памяти. Это позволяет написать программу процедуры обработки строки таблицы. При повторном вызове процедуры программу обработки строки таблицы необходимо привязать уже к другим адресам, в которых размещается другая строка таблицы. При этом существует запрет на изменения в теле программы в ходе её выполнения. (В противном случае будет утерян принцип стереотипности и повторяемости программы). Проблема перемещаемости процедуры над данными решается через механизм косвенной адресации. При косвенной адресации в адресных полях команды не указывается непосредственный адрес размещения операнда, а указывается некий фиксированный в памяти регистр, в котором находится реальный адрес операнда. Далее по этому адресу осуществляется обращение в память за нужным операндом. При косвенной адресации для доступа к операнду необходимо осуществить два цикла обращения в память. Сначала обратиться в фиксированный регистр, извлечь, содержащийся в нём адрес и далее по этому адресу извлечь операнд. Известно, что время исполнения команды в основном определяется временем цикла обращения в память. Следовательно, полученные нами 20% КПД необходимо снизить вдвое до 10%. И это ещё не всё. Для реализации перемещаемости процедуры над данными необходимо перед началом каждого следующего

прохода в цикле заменить адреса в фиксированных регистрах. Это осуществляется за счёт приращения адресов на некий фиксированный шаг. Таким образом, возникает целый пласт вычислительных работ, который называется адресная арифметика. Объёмы преобразований адресной арифметики равны объёмам основной обработки, поскольку необходимо модифицировать адреса всех операндов. Следовательно, КПД урезается ещё вдвое до 5%.

Из структуры команды, приведенной на рис. 4 следует, что возможности прямой адресации памяти из адресных полей команды весьма ограниченные. Если длина машинного слова равна 32 разрядам, каждое поле команды содержит по 8 разрядов. Это значит, что непосредственно из адресных полей команды можно адресовать не более чем 256 слов. По этой причине трёхадресный формат команды оказался проблемным и наибольшее распространение получил двухадресный формат, который позволяет при 32-х разрядном слове иметь два адресных поля по 12 разрядов и непосредственно адресовать 4096 слов. А это означает, что для реализации элементарной процедуры обработки в общем случае понадобится более одной команды, что только ухудшает временной баланс издержек. С учётом некоторой статистики применения коротких и длинных форматов команд мы вынуждены снизить КПД до 3%.

В современном компьютере адресное пространство оперативной памяти исчисляется гигабайтами. Дальнейшее расширение объёмов оперативной памяти стало возможным при осуществлении механизма многоступенчатой адресации. При ступенчатой адресации, например при двухступенчатой, пространство памяти расчленяется на страницы, регистры которых доступны из адресного поля команды. Страницам, в свою очередь, присваиваются свои адреса или номера страниц. Для реального доступа в память необходимо совместить адресное поле команды и номер страницы. При этом номер страницы представляет старшую группу разрядов, а адресное поле команды младшую группу разрядов реального адреса обращения в память. Номер страницы хранится в специальном индексном регистре в составе процессора. Это так называемый блок регистров общего назначения, сохраняющих вычислительную обстановку процессора и в частности его позиционирование в общем поле памяти. Назначение номера страницы и его занесение в индексный регистр осуществляет загрузчик, специальная системная программа низкого уровня, осуществляющая управление ресурсами памяти и обслуживающая загрузку приложения в память. Мы не станем здесь излагать детали этого достаточно громоздкого механизма, а только вынуждены будем констатировать, что осуществление перемещаемости приложения в памяти большого объёма и механизм многоступенчатой адресации урезают оценку КПД ещё наполовину до 1,5%.

При оценке эффективности организации вычислений, приведенной выше, мы ограничились рассмотрением уровня простейшей процедуры, погружённой в аппаратную среду. Это заведомо очень упрощённая ситуация. На самом деле необходимо понимать, что реальные условия работы компьютера осуществляются в системной среде, поддерживающей мультизадачные режимы. Выполнение системных функций и обработка прерываний ложатся на тот же единственный процессор и продолжают нагружать временную диаграмму. Так что в реальности эффективность функционирования современного классического компьютера по нашей экспертной оценке не превышает одного процента. В качестве метафоры можно утверждать, что на одно полезное действие приходится не менее сотни обеспечивающих, которые являются издержками принятых форм организации процессов.

Традиция проектирования аппаратных средств вычислительной техники складывалась в условиях подавляющей высокой стоимости аппаратуры. Поэтому издержки укладывались в последовательные ряды и выполнялись на одном оборудовании последовательно. По мере развития технологии микроэлектронного производства принцип экономии аппаратных затрат перестал быть доминирующим. Современные инженерные решения создают другую ситуацию, отличную от той на которой мы провели вышеприведенную экспертную оценку эффективности функционирования компьютера. Так, например, в современных процессорах программы и данные размещаются в разных блоках памяти, а выборки команд и данных осуществляются по разным каналам и совмещаются во времени, что заметно улучшает временную диаграмму. Текущие операнды подкачиваются порциями в сверхоперативную память и обеспечивают бесперебойную работу арифметического блока. И таких решений множество. Суть их сводится к тому, что издержки выносятся из последовательных рядов временной диаграммы и размещаются в дополнительной аппаратуре, которая работает параллельно с основной. Но главное заключается в том, что груз издержек остаётся прежним. Издержки не устраняются, а поглощаются дополнительной аппаратурой, а аппаратура это и есть основной ресурс кристалла, выраженный в числе транзисторов. Технология обеспечивает размещение миллиарда транзисторов на кристалле, и есть две стратегии его заполнения. Можно сохранить примитивную структуру процессора с последовательным размещением издержек во времени и построить кристалл, содержащий 100 ядер. А можно усложнить процессор разгрузить его временную диаграмму и разместить издержки на дополнительной аппаратуре. Но тогда на кристалле будет размещаться не 100 низкоскоростных ядер, а 20 высокоскоростных. Суммарная пропускная способность кристалла останется прежней, поскольку определяющим является факт существования сотни обеспечивающих действий на одно полезное. Это константа, характеризующая архитектурную идею. Именно введение критерия эффективности по соотношению числа полезных действий на единицу затрат энергии позволяет осуществить объективную оценку эффективности инженерных решений. Вынос обеспечивающих действий из временной диаграммы и размещение их на дополнительной аппаратуре создаёт иллюзию повышения эффективности при условии, что стоимость транзисторов ничтожна. Но энергию эти действия потребляют при любом размещении и при разных инженерных решениях никуда не деваются. Выделенные нами при экспертной оценке накладные потери являются следствием архитектурной идеи и порождаемых ею форм организации процессов. Инженерные ухищрения не устраняют издержки, а манипулируют их размещением, и только вновь введенный критерий энергоэффективности кристалла позволит запустить тенденцию к критической оценке и пересмотру архитектурной концепции.

Начинать работы по созданию высокопараллельных структур с таким грузом издержек и такой низкой эффективностью организации процессов, по меньшей мере проблематично, а по результатам, рассмотренных нами в первой части статьи примеров, бесперспективно. При распараллеливании издержки тиражируются кратно степени параллелизма. Но при этом неизбежно возникают новые специфические издержки на обслуживание параллелизма. Мероприятия по организации параллельной работы множества процессорных элементов несут в себе тот же порок низкоэффективных форм организации процессов. Т.е. при обслуживании параллелизма вновь генерируются десятки и сотни накладных расходных событий на единицы полезных. Поэтому при решении целого ряда актуальных задач с высоким потенциалом параллелизма насыщение роста

производительности наступает практически сразу при 10 – 15 процессорах и значениях 3-х, 4-х кратного ускорения. Параллелизм блокируется низкоэффективными формами организации вычислительных процессов. Факт существования эффекта насыщения не является фатальным. Точку насыщения можно перемещать и в перспективе обеспечивать реализацию высоких значений параллелизма. Фатальной является крайне низкая эффективность организации процессов в классической архитектуре.

Существует ли альтернативные решения, позволяющие строить вычислительные системы с более высокими показателями эффективности организации вычислительных процессов? Имеются ли приемлемые решения в области разработки неклассических архитектур? Наиболее значительным событием в этом направлении было создание архитектуры потока данных, известной как архитектура Data Flow. Первые публикации по архитектуре потока данных [10] были восприняты с большим энтузиазмом. Машина потока данных действительно представляет собой принципиально новый архитектурный проект, поскольку язык машины потока данных существенно отличается от линейного императивного языка машины фон Неймана. Перечислим эти отличия. Язык классической машины описывает программу как линейную последовательность команд. Команда представляет фиксированную элементарную процедуру вычислительного процесса. Адресные поля команды ссылаются не на данные, а на адреса регистров памяти. Команды исполняются в порядке их следования в записи программы. Здесь следует вновь напомнить, что внутренний язык классической машины не является языком описания вычислительных процессов, он описывает процессы изменения состояний памяти. Подгон процедур изменения состояний памяти к вычислительному процессу есть основное содержание работы программиста, но это содержание не фиксируется средствами языка машины. Это есть источник некорректности технологии программирования классических машин. А главный порок классической архитектуры это антагонизм последовательной записи процесса с его параллельной и асинхронной природой.

Основные положения архитектуры Data Flow мы уже излагали в первой части настоящей статьи [9]. Однако требование удобства чтения текста обязывают нас изложить эти положения повторно. Язык машины потока данных существенно отличается от языка классической машины по всем перечисленным позициям. Основной лексической единицей языка машины потока данных является пакет. Структура пакета приведена на рис. 5.

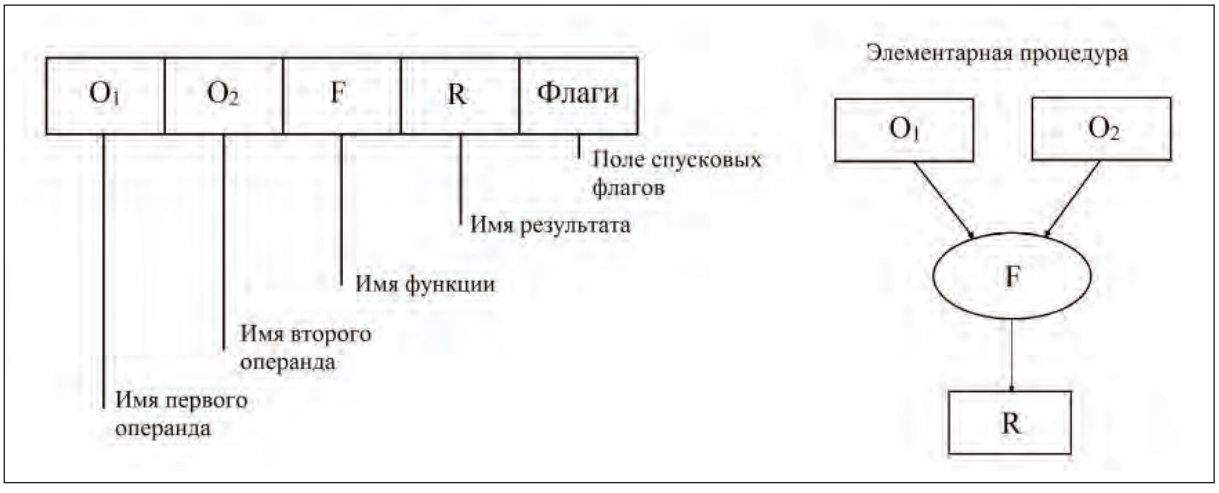


Рис. 5. Структура Data Flow пакета

Data Flow пакет также как и команда описывает элементарную процедуру вычислительного процесса, но в полях пакета записываются имена данных и это делает его языком описания вычислений. Элементарная процедура, описанная пакетом, не фиксирована, как в системе команд классической машины. Принимается гибкая система обозначений, при которой функция пакета может быть машинной командой или программой процедуры. Данные могут быть элементарными операндами или структурами данных. Такая гибкость очень важна при программировании параллелизма. Пакеты размещаются в памяти пакетов в произвольном порядке. В совокупности они образуют запись графа вычислительного процесса. Связность графа поддерживается взаимными ссылками имён аргументов и результатов. Память данных в потоковой машине является многофункциональной. Память данных осуществляет доступ к данным по именам, ведёт учёт их наличия и взаимодействует с памятью пакетов путём посылки фишек с именами поступивших данных. Фишки, принятые в памяти пакетов инициируют операции над спусковыми флагами. В ситуации, когда спусковые флаги пакета сигнализируют о наличии поименованных аргументов, пакет переводится в активное состояние и выставляет маркер готовности. В каждый текущий момент времени в памяти пакетов образуется множество готовых пакетов, которые отправляются на обработку, а в результате их обработки порождаются новые данные, которые спускают на обработку новые порции пакетов. Логика функционирования знаковой системы (языка) машины потока данных не требует специальных мер программирования параллелизма. Знаковая система самостоятельно извлекает динамический параллелизм из текущего состояния данных.

Перечисленные качественные характеристики архитектурной идеи, лежащей в основе проекта Data flow, воспринимаются как более предпочтительные для построения систем с массовым параллелизмом. Однако до настоящего времени неизвестно ни одного варианта их промышленной реализации - проект Data flow не выходит из стадии исследования и разработки. Наш опыт модельных испытаний системы с элементами Data flow, описанный в первой части настоящей статьи даёт тот же результат быстрого прекращения роста производительности на уровне 3-х кратного ускорения при 6 процессорах. Для того, что бы нащупать путь выхода из этого тупика необходимо понять, что объединяет оба проекта, несмотря на обозначенные нами существенные их различия. Общим для классической архитектуры и для проекта Data flow является принцип опережающей разметки трассы процесса. В обоих случаях знаковая система, лежащая в основе архитектуры, является средством создания символьной копии процесса. В классической машине символьная копия процесса существует в виде последовательности команд, а в архитектуре Data flow в виде потокового графа. В классической архитектуре команды считываются синхронно по одной в порядке следования в записи, а в Data flow активируются асинхронно группами в соответствии с логикой спуска по готовности данных. Но и в том и в другом случае активированные команды и пакеты не являются реальными агентами осуществления процессов обработки и вычислений, это только ссылки на данные и на функции. В след за активацией фрагментов трассы процесса разворачиваются потоки обеспечивающих событий, необходимых для поиска, доступа в память, чтения и транспорта данных к местам локализации функций обработки, а за тем по транспорту и размещению в памяти результатов обработки. Основным источником многочисленных накладных потерь являются работы по обслуживанию трассы процесса. Причём, запись трассы в виде потокового графа в Data flow в от-

личие от линейной последовательности в классической машине, порождает более сложные и объёмные работы по её обслуживанию. Проблема многократного использования стереотипных фрагментов трассы при линейной последовательной записи требует выделения двух точек записи – начала и конца процедуры. Для многократного повторного использования фрагмента потокового графа необходимо обозначить множество входных и выходных полюсов, привязанных к соответствующим вершинам графа. Далее необходимо организовать очереди фишек на входных полюсах, а сами фишки разметить по принадлежности к поколениям данных, возникающим в потоке обработки. Эта проблема получила название задачи раскраски фишек. А затем решать проблему синхронизации потоков фишек на всех входных полюсах процедуры. И всё это ещё не процесс вычислений – это только работы по обслуживанию записи трассы процесса. И это далеко не единственная проблема проекта, но она позволяет понять, почему проект Data flow до сих пор остаётся проектом.

Поищем аналогии в другой области. Похожая ситуация имела место на начальном этапе разработки проекта глобальной сети связи в конце 60-х годов [11]. Географически распределённые сети связи континентального и межконтинентального масштаба управлялись централизованно через центры коммутации. Сети функционировали на базе технологии коммутации каналов. Для осуществления сеанса связи через центры коммутации прокладывался маршрут, который монополизировал дорогостоящие магистральные ресурсы на время сеанса для пары абонентов. Для быстрого обслуживания плотных потоков требований на соединение приходилось наращивать число параллельно работающих каналов. Когда проектировщики задумались о построении глобальной единой автоматизированной системы связи (ЕАСС) и стали подсчитывать необходимые объёмы капитальных вложений оказалось, что проект по стоимости неподъёмный для самых богатых стран и их совместных усилий. Тогда инженеры связисты сделали экспертные оценки эффективности использования магистралей связи. Оказалось, что эффективность их загрузки чудовищно низкая и лежит в пределах нескольких процентов. Существовала проблема выбора - или строить и вводить в эксплуатацию гигантские объёмы новых магистралей или решить системную проблему и поискать более продуктивные технологии связи с лучшими показателями эффективности. Результат нам известен - была создана технология коммутации пакетов, современная глобальная паутина это сеть пакетной коммутации [12].

В пакетной сети отсутствует централизованное управление и коммутация каналов. Сети равномерно покрывают территории как связные совокупности узлов, где каждый узел связан со смежными узлами, каналами открытого общего доступа. Информация, циркулирующая в каналах, имеет цифровое представление. Сообщения фрагментируются на относительно небольшие порции, которые оформляются в сетевые пакеты. Сетевой пакет это формализованная запись, в которой есть блок фиксации содержательной информации и ряд сопровождающих функциональных полей. Содержательная информация может быть графической, текстовой или акустической, но сеть к этому безразлична. Интерпретация содержательной информации осуществляется терминальным оборудованием, принадлежащим абонентам. Сеть становится унифицированной. Пакеты порождаются терминальным оборудованием пользователей, большими массами вбрасываются в сеть и начинают самостоятельно продвигаться по сети от узла к узлу. Ресурсы сети никогда не монополизируются отдельными сеансами связи и всегда открыты для доступа. Алгоритмы обработки пакетов в узлах построены таким образом,

что пакет может самостоятельно выбирать маршрут и прокладывать свой путь к заданному адресату. Вся необходимая управляющая информация содержится в функциональных полях пакета, сопровождающих поле содержательной информации. В узлах имеются полные каталоги маршрутов и кроме того узлы периодически обмениваются между собой служебными сообщениями, в которых обозначают свою текущую загрузку. Таким образом, пакет на каждом шаге продвижения от узла к узлу уточняет свой маршрут и имеет возможность обхода перегруженных участков сети. В результате сетевая технология пакетной коммутации позволила увеличить продуктивность сетей и эффективность использования магистральных ресурсов на порядки.

Подводя итог обсуждения проблемы эффективности методов организации вычислений можно отметить, что необходимость существенного обновления архитектурной идеи вполне назрела и контуры новой архитектуры определились. Далее мы сформулируем перечень основополагающих принципов новой архитектурной концепции.

Отказ от принципа раздельного существования потоков команд и потоков данных. Должен функционировать один поток - поток данных, сопровождаемых управляющими функциональными полями по аналогии с организацией сетевых пакетов в сетях пакетной коммутации. Основные лексические элементы знаковой системы, организующей вычислительный процесс должны представлять собой самоопределяемые операнды. Самоопределяемый операнд состоит из поля, несущего содержательную информацию, и набора сопровождающих функциональных полей, несущих всю управляющую информацию, детерминирующую поведение и виды активности самоопределяемого операнда.

Отказ от принципа опережающей разметки трассы процесса как следствие отказа от потока команд, которые и были средством формирования символьной копии процесса. По сути это отказ от принципа сосредоточенного управления процессом. Символьная копия процесса отсутствует, а вся управляющая информация распределена по непосредственным участникам процесса – самоопределяемым операндам. Трасса процесса теперь не актуализована в записи программы, а проявляется косвенно в динамике вычислений как результат наблюдения за процессом. Это и есть принцип распределённого управления.

Отказ от технологии коммутации каналов означает, что обмен данными не должен существовать как отдельная операция, связывающая параллельные фрагменты процесса. При сосредоточенном управлении процессами параллелизм порождает необходимость обмена данными как отдельный компонент процесса, требующий отдельной физической обменной среды. При этом обменные операции, порождают накладные затраты, растущие с ростом параллелизма. В новой архитектуре принимается принцип сетевой организации вычислений, при которой движение данных совмещается с процессом обработки. Практически это означает устранение значительных объёмов паразитных пересылок данных и максимальный параллелизм необходимых обменов данными на низовом уровне, не требующем специальных коммутационных сред.

Перечисленные принципы соответствуют тенденциям, которые фактически сложились в практике построения многопроцессорных систем. Наиболее полное воплощение сетевых принципов организации процессов просматривается в структурах с топологией тора [3, 4, 5]. В тороидальных топологиях применяется принцип пакетной маршрутизации данных и главное заключается в том, что в узлах сети размещаются процессорные элементы, а в целом узлы сети совмещают функции обработки и коммутации данных.

Осталось сделать лишь завершающее усилие – расширить функции распределённого управления маршрутизацией пакетов до функций управления вычислениями и далее распространить сетевые принципы управления на все уровни организации вычислений.

3. Архитектура самоопределяемых данных и принципы распределённого управления

Идея построения архитектуры самоопределяемых данных имеет определённую предысторию, некоторые её фрагменты изложены в ряде публикаций и патентов прошедших лет [13, 14, 15]. Однако в прошлые десятилетия идея не воспринималась как конкурентоспособная. Технология позволяла удовлетворять требования закона Мура и наращивать производительность вычислительных средств исключительно за счёт роста тактовой частоты и усложнения процессорного элемента. Кроме того смена принципов организации вычислений нарушала требование преемственности и переносимости, наработанных ранее программных продуктов и резкая смена архитектуры была экономически не целесообразна. Но в настоящее время, когда реализация массового динамического параллелизма становится основным направлением развития вычислительных средств, старый проект может стать востребованным и занять определённую нишу в существующем разнообразии областей применения.

Рассмотрим базовые принципы функционирования архитектуры самоопределяемых данных. Основным элементом знаковой системы является самоопределяемый операнд. Это упорядоченная запись, состоящая из содержательного поля, в котором размещается операнд и нескольких сопровождающих функциональных битовых полей. Формат самоопределяемого операнда приведен на рис. 6.

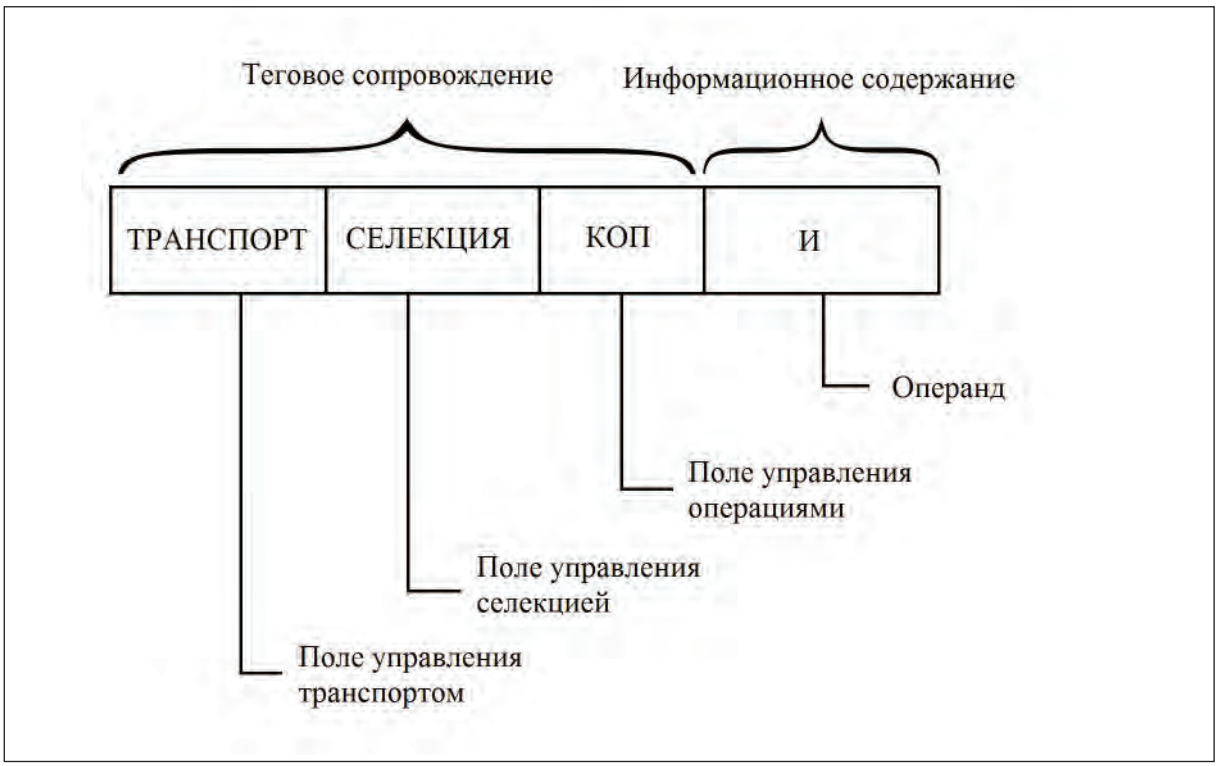


Рис. 6. Формат самоопределяемого операнда

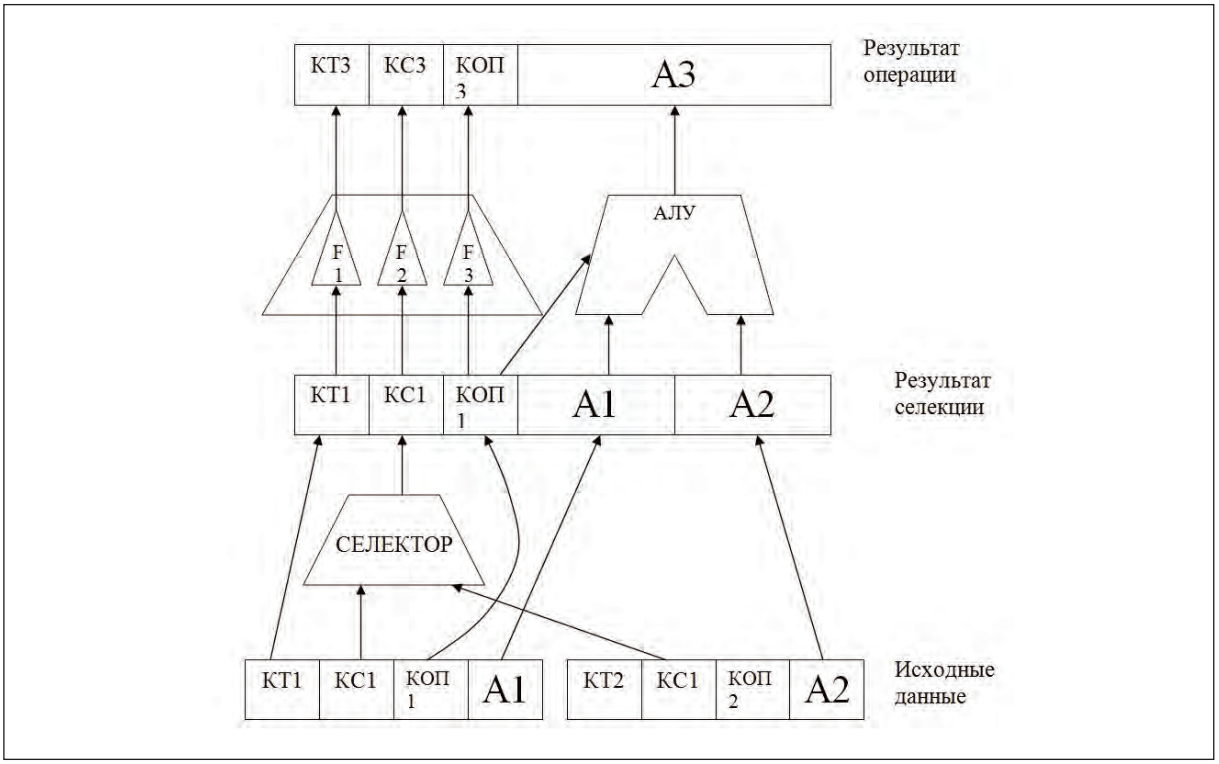


Рис. 7. Схема взаимодействия самоопределяемых операндов

Набор функциональных полей называется теговым сопровождением и включает в себя поле управления операциями, поле управления селекцией пар операндов и поле управления транспортом. Кроме перечисленных функциональных полей в теговом сопровождении может быть предусмотрено резервное поле, предназначенное для оперативного расширения управляющих функций в случае необходимости. Это означает, что набор управляющих полей открыт и может корректироваться по текущей обстановке. В приведенном формате рассматривается минимальный набор управляющих полей. Смысл управления состоит в том, что самоопределяемый операнд должен осуществлять селекцию пары для выполнения бинарной операции, определять тип операции обработки и принимать решения о перемещении по транспортным путям вычислительного устройства.

Рассмотрим логику взаимодействия самоопределяемых операндов проиллюстрированную диаграммой на рис. 7.

В нижней части схемы изображены два операнда, предназначенные для выполнения заданной бинарной операции. Операция селекции может осуществляться аппаратно или программно и смысл селекции состоит в том, что селектор отбирает пары операндов по кодовым состояниям полей селекции. На выходе селектора формируется блок, называемый парой. Пара состоит из двух информационных полей A_1 и A_2, предназначенных для выполнения бинарной операции. К паре пристыковывается теговое сопровождение одного из операндов, второй тег после селекции отбрасывается. Отселектированная пара направляется в канал обработки, который состоит их двух секций – арифметического блока и преобразователя теговых кодов. Арифметический блок это стандартное арифметико-логическое устройство (АЛУ), которое управляется кодом операции из тегового сопровождения пары. Секция преобразования тегов содержит для

каждого тегового поля свой унарный функциональный преобразователь. В результате прохождения через канал обработки образуется новый операнд A_3 с новым теговым сопровождением.

Вычислительное устройство состоит из множества каналов обработки и путей перемещения, в которые загружается множество исходных операндов. Поведение исходных операндов определяется кодами их теговых сопровождений. В результате реализации предписанных тегами событий порождается второе поколение операндов с новыми кодами тегов, определяющими следующий шаг обработки.

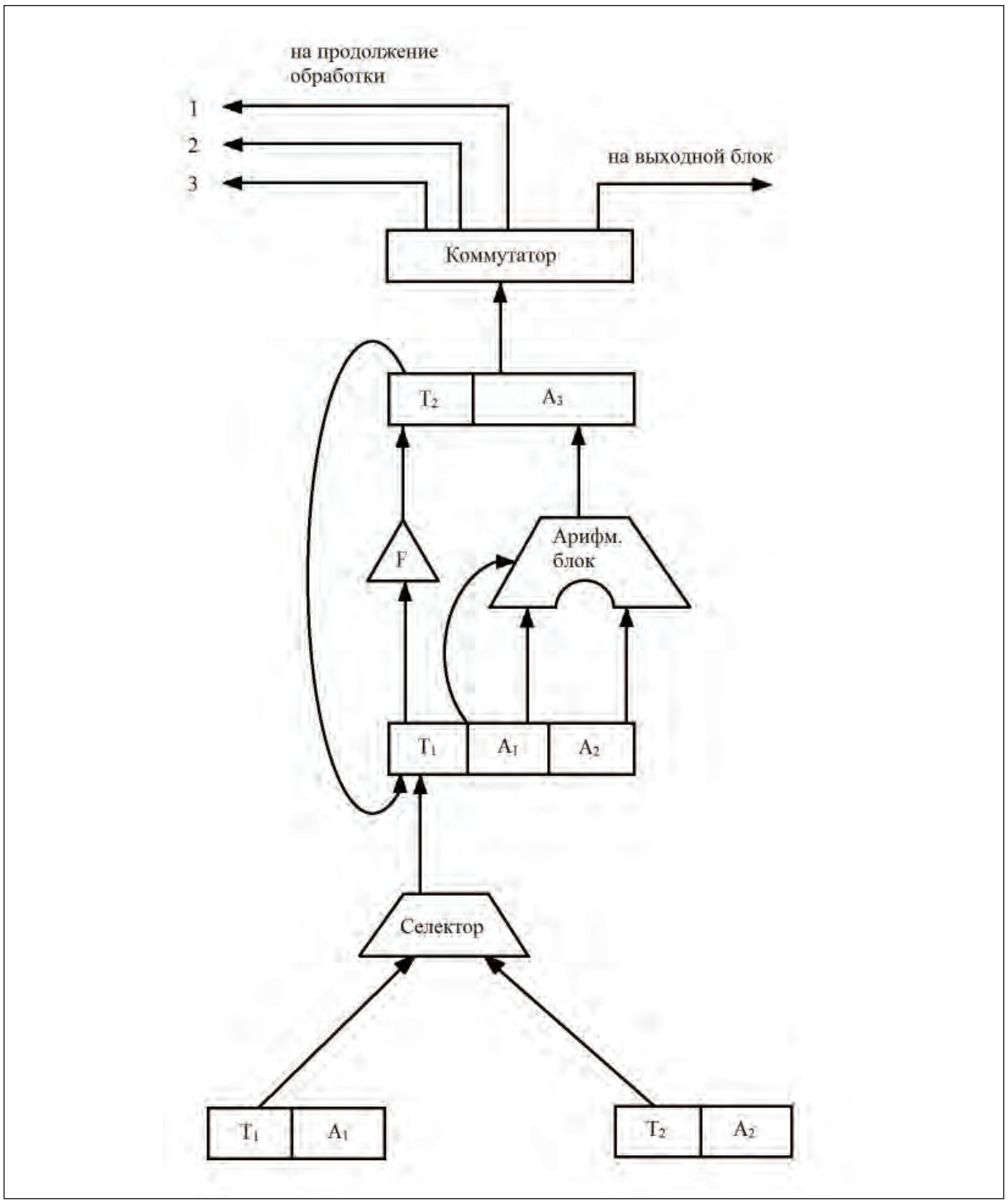


Рис. 8. Схема взаимодействия упрощённых самоопределяемых операндов

Для более детального рассмотрения динамики развития процесса обработки введём упрощённый вариант построения самоопределяемых операндов. Будем считать, что селекция пар операндов осуществляется по совпадению кодов селекции. Это даёт возможность совместить в одном поле управление селекцией и кодирование операции, поскольку для селекции важен только факт совпадения кодовых значений пары селектируемых операндов. Также с целью упрощения ситуации можно опустить поле управления транспортом. В результате мы станем рассматривать упрощённый самоопределяемый операнд с одним теговым кодом. Схема взаимодействия таких упрощённых операндов изображена на рис. 8.

Селектор вылавливает в потоке самоопределяемых данных операнды с совпадающими значениями теговых кодов и формирует из них пару. Пара операндов поступает в канал обработки, где над информационными полями выполняются арифметические операции, а теговые коды проходят через дискретный функциональный преобразователь. На выходе канала обработки формируется новый операнд с новым значением тега. На выходе канала обработки изображён коммутатор, в котором определяется выбор направления передачи операнда. Дешифрирующая схема коммутатора анализирует определённые разряды тега и принимает решение о перемещении операнда, либо в выходной блок для формирования выходных данных, либо в одно из трёх направлений продолжения обработки. Приведенная схема иллюстрирует, как именно самоопределяемый операнд прокладывает путь своих перемещений в аппаратной среде.

Далее новые поколения операндов вновь вылавливаются селекторами и передаются в каналы обработки, где теговые коды вновь проходят через функциональный преобразователь. Путь превращений теговых кодов выделен на схеме дугообразной стрелкой, замыкающей вход и выход функционального преобразователя. Динамика развития вычислительного процесса осуществляется следующим образом. В секции преобразования теговых кодов всех каналов обработки загружается определённая функция **F**. В вычислительное устройство вводится набор исходных операндов, с заданными значениями тегов. В силу принятых принципов построения аппаратуры из исходного набора операндов селекторами отбирается множество пар, которые загружаются в каналы обработки для выполнения бинарных арифметических операций. Таким образом осуществляется первый шаг обработки, который полностью определяется состояниями теговых кодов, присвоенных исходному набору операндов. В результате выполнения первого шага на выходе каналов обработки формируется второе поколение операндов с новыми значениями тегов. Далее поведение вновь образованных операндов полностью определяется значениями новых теговых кодов и таким образом осуществляется следующий шаг. Динамическим ядром процесса обработки является рекуррентный генератор кодовых последовательностей, который просматривается на схеме рис. 8. Рекуррентный генератор работает в режиме самообращения и на каждом шаге преобразования передаёт выходной теговый код на вход преобразователя тегов F в качестве нового аргумента.

В классической машине динамика вычислений формируется в результате последовательного считывания программы, которая представляет собой статическую запись опережающей разметки трассы процесса. В самоопределяемых данных процесс разворачивается динамически шаг за шагом и задаётся в аналитической форме в виде функции преобразования теговых кодов **F**. При этом важно отметить, что в данном случае аналитическое представление динамики процесса не является функцией от времени, а является функцией от предыдущего состояния. В архитектуре самоопределяемых дан-

ных роль программы как детерминанта процесса выполняет функция преобразования тегов **F** и начальные состояния теговых кодов в исходном наборе операндов.

Можно ли в описанной логике взаимодействия самоопределяемых операндов сконструировать осмысленный вычислительный процесс? Рассмотрим конкретный пример на базе введенного ранее упрощённого представления самоопределяемых операндов с одним теговым полем, совмещающим кодирование селекции пар и арифметических операций. Для простоты и наглядности примера будем считать, что тег имеет минимальную разрядность три бита, а функция дискретного кодового преобразователя **F** есть простая операция сдвига битового набора вправо на один разряд. Трёхразрядный битовый код имеет всего восемь возможных состояний и заданное отображение **F** можно представить в виде таблицы. Отображение **F** также можно представить в виде графа, в котором кодовые состояния это вершины графа, а рёбра определяются как пары ***x; F(x).*** Назовёт такой граф графом кодовых переходов. Таблица отображения **F** и граф кодовых переходов приведены на рис. 9. На графе кодовых переходов для удобства чтения вершины обозначены как целые десятичные числа. Необходимо также назначить кодовым значениям тега арифметические операции и зафиксировать соответствие в виде таблицы, по которой арифметический блок будет интерпретировать теговые коды. Таблица кодирования арифметических операций также приведена на рис. 9. Поскольку кодовых состояний 8, а арифметических операций только 4, целесообразно привязать лишние коды к имеющимся операциям повторно. В данном случае соответствие не должно быть обязательно взаимно однозначным. Код «0» присваивается операции завершения процедуры и интерпретируется аппаратурой как останов.

И так мы сделали все необходимые определения и можем проследить как будет развиваться вычислительный процесс. Графическое изображение процесса в виде графовой диаграммы приведено на рис. 10. Прямоугольниками на диаграмме обозначены операнды, состоящие из двух полей – тега и информационного поля. Пирамидками обозначены процессы, включающие операции селекции пар, арифметические операции и операции преобразования тегов.

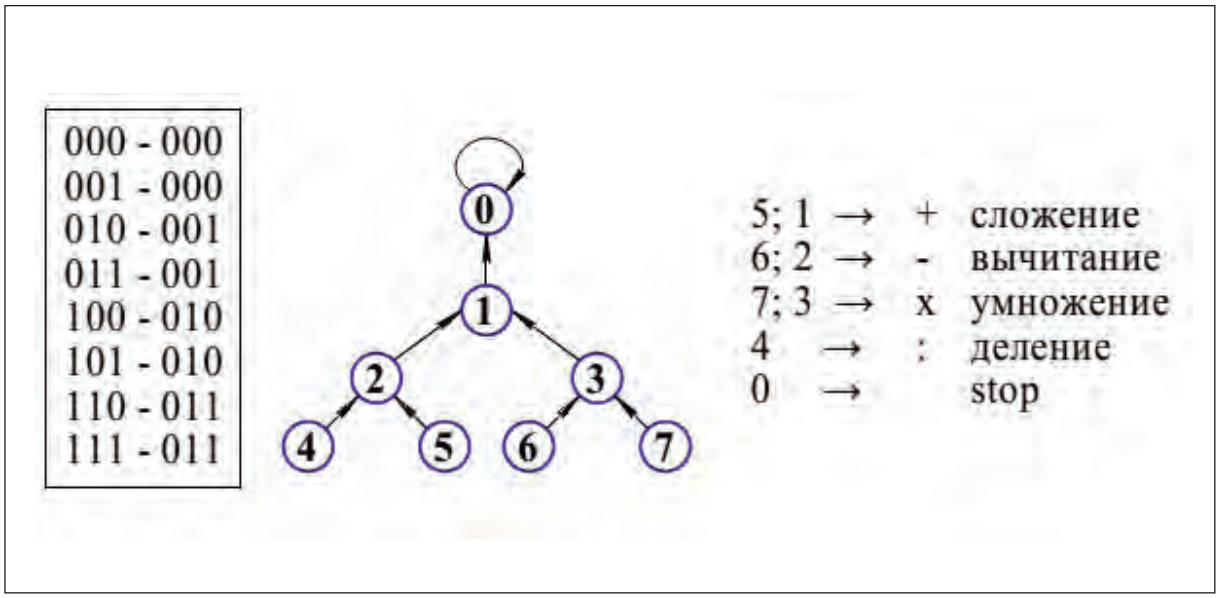


Рис. 9. Таблица отображения **F** и граф кодовых переходов

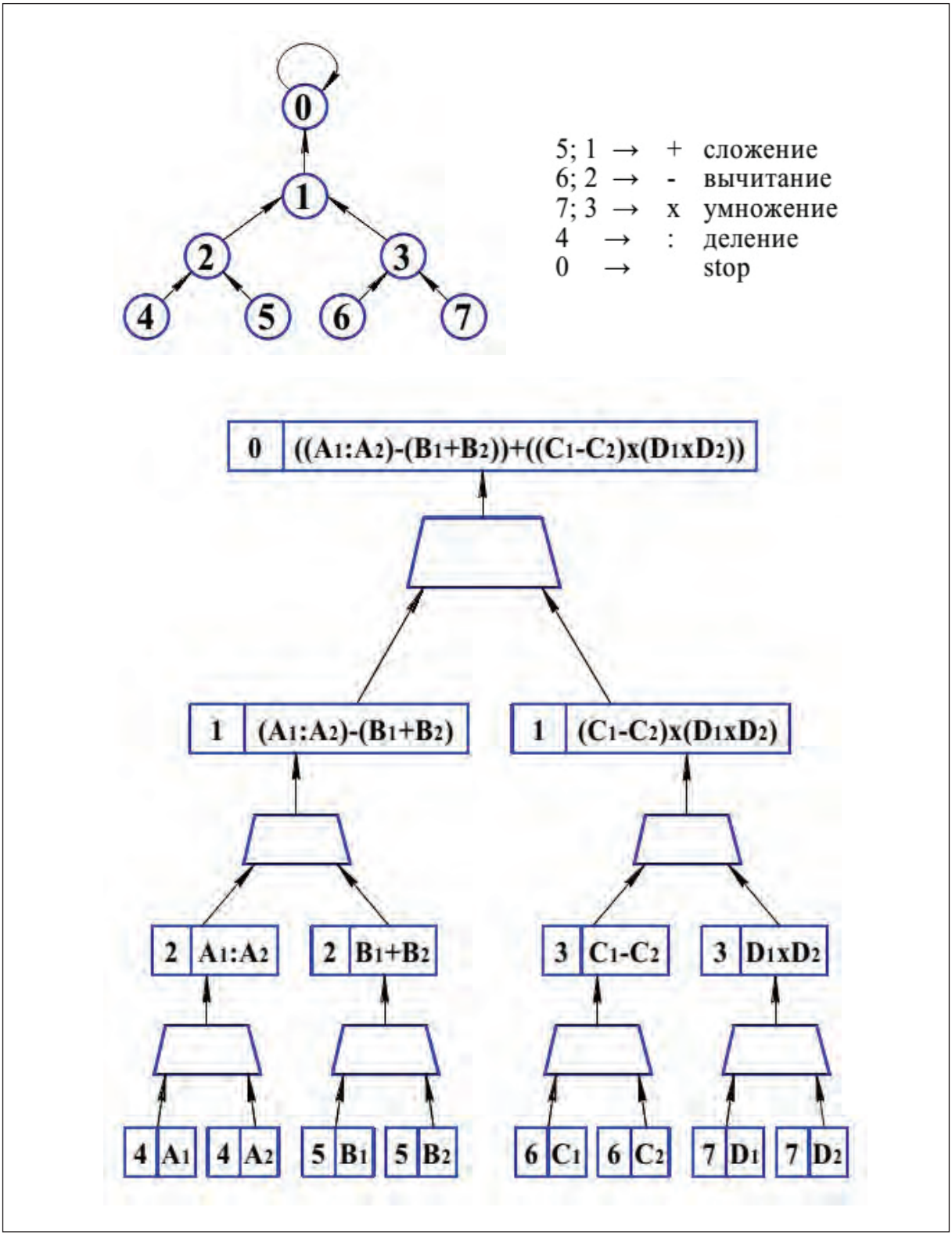


Рис. 10. Диаграмма вычислительного процесса

Перед началом работы во все каналы обработки вычислительного устройства загружаются преобразователи тегов **F** и таблицы интерпретации теговых кодов. Далее вводится исходный набор операндов. Исходные операнды заполняют нижний слой диаграммы на рис. 10. Теговые коды исходных операндов позволяют отселектирвать пары

и направить их в каналы обработки, что отображено во втором слое диаграммы. Третий слой диаграммы заполнен результатами выполнения первого шага обработки, представляющими второе поколение операндов. Формирование содержания полей операндов второго поколения можно проследить по графу кодовых переходов и по таблице интерпретации тегов, которые для наглядности приведены на рис 10. Преобразования тегов прослеживаются на графе кодовых переходов, а выполняемые арифметические операции определяются таблицей кодирования операций. В информационные поля операндов записывается операция вычисления нового операнда. Во втором поколении операндов в полном соответствии с графом кодовых переходов образуются пары с одинаковыми тегами, которые на следующем шаге обработки вновь отлавливаются селекторами и передаются в каналы обработки. И таким образом события разворачиваются до появления тега со значением «0», что интерпретируется аппаратурой как операция останова и вывода результата в выходной блок.

На данной иллюстрации при изображении операндов следующих поколений, отличных от исходных и порождаемых в процессе обработки, в информационное поле вместо символического обозначения операндов вписаны процедуры их вычисления. Это необходимо для иллюстрации поэтапной сборки заключительного арифметического выражения, описывающего всю наблюдаемую вычислительную процедуру.

Рассмотренный пример показывает, что при заданной логике взаимодействия операндов и некоторых произвольно выбранных начальных условиях в системе развивается детерминированный процесс, соответствующий вычислению определённого арифметического выражения. Параллелизм извлекается аппаратурой из текущего состояния данных и не требует специальных мер синхронизации, привязки данных к каналам обработки и других мер программирования и обслуживания параллелизма. При условии, что во все каналы обработки загружена одна функция преобразования тегов и одна таблица интерпретации тегов, любой операнд может обрабатываться в любом канале. Где операнд оказался в текущий момент, там он и обрабатывается. При этом привязка операнда к определённой позиции на графе процесса обработки содержится в его теговом сопровождении и неотделима от него. Таким образом, архитектура самоопределяемых данных ликвидирует накладные потери в виде многочисленных паразитных пересылок данных из мест хранения к местам обработки и обратно.

Для наглядности и простоты примера мы выбрали малое значение разрядности тега, равное трём. Но даже в этом простейшем случае вычисляемое арифметическое выражение оказалось относительно протяжённым и не тривиальным. Достаточно увеличить теговый код на один разряд и число вершин в графе кодовых переходов удвоится, соответственно удвоится и сложность вычислительной процедуры. При 8-разрядном теге граф кодовых переходов с топологией бинарного дерева будет состоять из 256 вершин и 8 этажей, что позволит поддерживать достаточно сложные и многоэлементные вычислительные процедуры. При этом затраты на программирование и поддержку программы (вернее того, что является аналогом программы) останутся незначительными. В классической архитектуре сложность программирования и сложность алгоритма связаны очень просто. Для простой процедуры надо написать десять строк программы, а для сложной тысячу строк. А в самоопределяемых данных для усложнения программируемой процедуры на порядок достаточно только увеличить разрядность тегового кода примерно на три – четыре разряда.

В предыдущем разделе при оценке эффективности разных форм организации вычислений рассматривались механизмы, поддерживающие многократное использование стереотипных фрагментов программы или так называемых стандартных процедур. Было показано, что как в классической архитектуре, так и в архитектуре Data Flow обеспечение перемещаемости процедуры и привязка тела программы процедуры к разным данным является источником огромных издержек и в программном и в аппаратном обеспечении. Посмотрим как эта проблема решается в архитектуре самоопределяемых данных. Рассмотрим тот же пример с обработкой табличных данных, в котором строка таблицы содержит десяток атрибутов, а таблица в целом состоит из нескольких сотен строк. Создаётся процедура обработки строки, для неё формируется функция преобразования тегов **F** и набор теговых кодов, присваиваемых исходным операндам из строки таблицы. Для тиражирования процедуры и привязки её к разным строкам таблицы достаточно ввести в теговое сопровождение операндов дополнительное поле индексной метки строк таблицы. (В первичных определениях констатировалось, что набор теговых полей открыт для пополнения и введение новых полей не нарушает основы архитектуры и не требует перестраивать конструкцию машины). Тогда все операнды, принадлежащие одной строке таблицы, будут иметь одно значение в индексном поле, а разные строки будут иметь разные индексные метки. Поле индексной метки будет участвовать в операции селекции, но не будет участвовать в преобразовании теговых кодов. Далее можно загружать таблицу в вычислительную среду в произвольном темпе и в произвольном порядке. Операнды, принадлежащие одной строке всегда отселектируются в один процесс обработки строки, а атрибуты разных строк никогда не пересекутся на всех стадиях выполнении процедуры. Любая процедура в архитектуре самоопределяемых данных может быть тиражирована с заданной кратностью путём расширения тегового сопровождения. Дополнительные разряды тега это необходимая плата за обеспечение требуемого свойства. Можно предполагать, что в данном случае издержки окажутся ощутимо меньше чем в классической архитектуре.

В рассмотренном примере реализации процедуры вычисления арифметического выражения функция преобразования тегов, которая является динамическим ядром процесса, порождала граф кодовых переходов с топологией бинарного дерева. Существует целый класс функций, заданных на конечных дискретных наборах и порождающих бинарные деревья разной размерности и с разным распределением кодов по вершинам графа. Понятно, что процесс вычисления арифметического выражения на базе двухместных операций имеет топологию бинарного дерева и может быть размещён на подходящем графе кодовых переходов, порождаемом функцией преобразования тегов. Означает ли это, что в архитектуре самоопределяемых данных можно реализовать только процессы с топологией бинарного дерева?

Рассмотрим возможности расширения разнообразия вычислительных процедур в самоопределяемых данных. Поскольку в машинной обработке приняты двухместные базовые операции, базовые процессы имеют топологию бинарного дерева. Но можно усложнить топологию реального процесса путём склеивания бинарных деревьев в определённых вершинах. Это потребует механизма многократного вхождения некоторых операндов в граф вычислений либо обмена операндами между разными поддеревьями. Создать набор непересекающихся бинарных деревьев можно при условии, что теговый код имеет достаточно большую разрядность и порождает большое дерево. Тогда при одной порождающей функции в графе кодовых переходов можно выделить набор

непересекающихся поддеревьев. Локализация поддеревьев определяется выбором листовых кодов. Для привязки групп исходных операндов к своим фрагментам процесса необходимо присвоить им в качестве тегов листовые коды выбранных поддеревьев. Многократное вхождение определённых операндов в граф вычислений можно обеспечить созданием в исходном наборе операндов ряда копий операнда с разными тегами, поскольку именно тег позиционирует место операнда на графе.

Широкие возможности многократного вхождения операндов в граф процесса обеспечивает некоторое усложнение операции селекции путём маскирования части разрядов теговых полей, в частности полей индексных меток. В этом случае возможен множественный отклик при селекции и организация векторной обработки.

Наиболее важным инструментом расширения динамики процессов в системе самоопределяемых данных является механизм операций с шаблонами. Это аналог команд перехода в классической машине. Команда перехода несёт в своём составе адрес новой точки в теле программы и нарушает естественный порядок следования команд путём замены состояния счётчика команд на адрес принудительного перехода. В самоопределяемых данных позиционирование операнда на графе процесса определяется теговым кодом. Смысл операции с шаблоном заключается в том, чтобы принудительно сменить теговый код операнда и перебазировать его в другой фрагмент графа процесса. Шаблон имеет тот же формат, что и самоопределяемый операнд и так же путём селекции находит своего партнёра, но в информационном поле шаблона вместо операнда записан новый тег для партнёра по селекции. При выполнении операции с шаблоном операнд в информационном поле не меняется, а получает новый тег из информационного поля шаблона. Операции с шаблоном позволяют организовать разнообразные условные и безусловные переходы, итеративные циклы и рекурсивные конструкции.

Рассмотренный пример показывает, что организация процессов в архитектуре самоопределяемых данных строится на базе двух компонент – динамического ядра и интерпретатора. Динамическое ядро порождает каркас процесса в виде графа кодовых переходов, а интерпретатор читает порождаемые коды теговых сопровождений операндов и придаёт им содержательное значение, например в виде арифметических операций. Архитектурная идея, не требуя изменений своей сути, допускает разнообразие интерпретаций. Информационные поля операндов могут содержать не только числа, а действия могут выполнять не только арифметические операции. Это могут быть символьные строки и текстовые объекты, это могут быть фрагменты графики или структур данных. Рассмотренный нами пример можно воспринимать как вычисление заданного значения, а можно считать его процессом сборки символической записи из базовых лексических единиц. Возможно значительное разнообразие применений архитектурной идеи самоопределяемых данных для построения реальных информационных систем.

Рассмотренный нами пример строился таким образом, чтобы продемонстрировать, что в принятой логике взаимодействия самоопределяемых операндов заданные начальные условия разворачиваются в определённый и осмысленный вычислительный процесс. Но это подтверждает принципиальную состоятельность выдвинутой архитектурной идеи только частично. Необходимо так же показать возможность решения обратной задачи – по заданному процессу построить начальные условия, которые смогут развернуться в этот процесс. Решение этой задачи означает возможность построения технологии программирования процессов в системе самоопределяемых данных. Из рассмотренного примера следует, что технология программирования самоопределяе-

мых данных существенно отличается от привычной технологии программирования классических машин. Суть этого отличия состоит в том, что эвристическое программирование в виде прямой записи символьной копии процесса без привлечения математических инструментов более невозможно. Для построения технологии программирования самоопределяемых данных требуется специальный математический аппарат описания процессов и целый ряд прикладных математических инструментов, разработанных на базе этого аппарата.

4. Основные понятия дискретной динамики

4.1. Работы В.И. Арнольда, основные принципы

Мы будем опираться на ряд фундаментальных работ В.И. Арнольда, которые тематически объединяются одним названием – исследование геометрии функциональных пространств, заданных на конечных множествах. Работы изложены в целом ряде публикаций [16, 17, 18].

Смысл этих работ заключается в том, что исследуется базовое понятие математики – структура, которая задаётся функциональным оператором на конечном множестве элементов. Основная масса исследований структур базируется на аналитической записи оператора и представляется как система манипуляции символическими конструкциями. Арнольд предложил представлять оператор и структуру в виде ориентированного графа. Графовое представление возможно лишь в тех случаях, когда множество, образующее структуру конечно. Граф структуры строится по правилу – элементы структуры являются вершинам графа, а рёбра определяются как пары ***x; F(x).*** Возможно только однократное вхождение элемента в граф. Арнольд назвал этот граф **монадой**. Этим архаичным названием подчёркивалось фундаментальное значение введенного понятия как древнего первоосновного. Традиция исследования монад восходит к Ньютону, Лейбницу и возможно имеет и более ранних предшественников [18]. Арнольд отмечает, что первоначально построение монад носило эмпирический характер и оказалось интригующим и увлекательным занятием. В последствии ученики и аспиранты применили компьютеры для построения монад и дело двинулось более высокими темпами.

Основные свойства монад достаточно очевидны и следуют из первичных определений. В монаде всегда есть хотя бы один цикл, это следует из конечности исходного множества. Поскольку оператор образующий структуру всюду определён, в некоторый момент для результата применения оператора не хватит элементов, и результат замкнётся на ранее использованных. Если граф монады содержит несколько циклов, множество распадается на непересекающиеся классы элементов, тяготеющих к своему циклу. Это происходит, потому что между циклами рёбер не может быть, в противном случае нарушается функциональность отображения ***F***. На рис. 11 приводится перечень возможных структур графов монад.

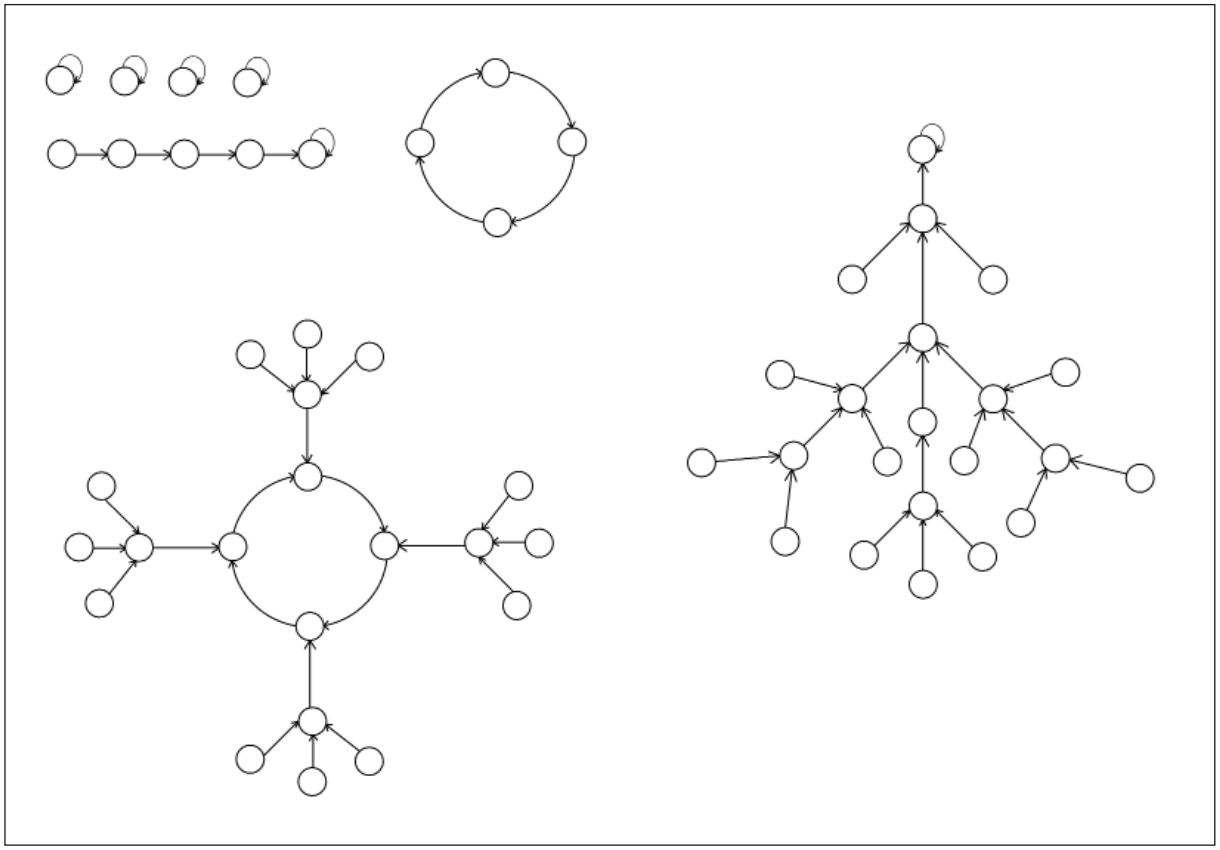
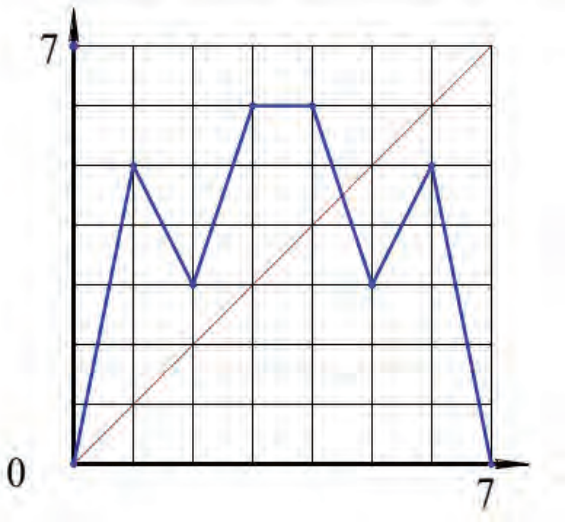


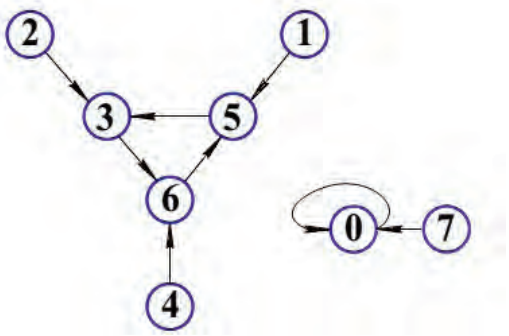
Рис. 11. Примеры графов монад

Множество элементов, тяготеющих к своему циклу, Арнольд назвал аттрактором. Исследования конечных структур, представленных монадами использовались Арнольдом для построения шкалы оценки сложности математических объектов, для исследований в теории чисел и ряде других направлений.

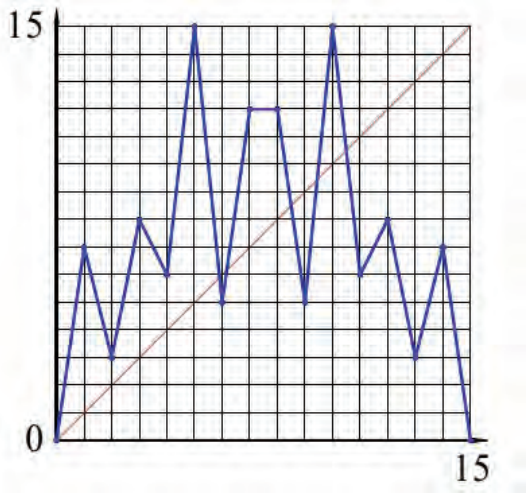
Представление структуры в виде монады позволяет увидеть тонкие закономерности и свойства, которые ускользают или маскируются при аналитической записи. Так, например, рассмотрим структуру, образованную дискретным функциональным преобразователем, заданным на множестве состояний компьютерного регистра. Оператор может представлять собой арифметическое соотношение или булеву функцию, составленную из набора побитовых булевых операций. Построение монад для данной структуры обнаруживает неожиданный факт - структура графа монады зависит от числа элементов образующего множества. При изменении разрядности регистров и неизменном функциональном операторе структура графа претерпевает значительные изменения. Пример иллюстрируется на рис. 12. Оператор **F** представляет собой следующую последовательность элементарных операций: над исходным операндом выполняется операция циклического сдвига на один разряд, далее полученный результат складывается с исходным операндом по модулю 2. На рис. 12 приводятся графики заданного отображения **F** при разных значениях разрядности регистров и соответствующие им графы монады.

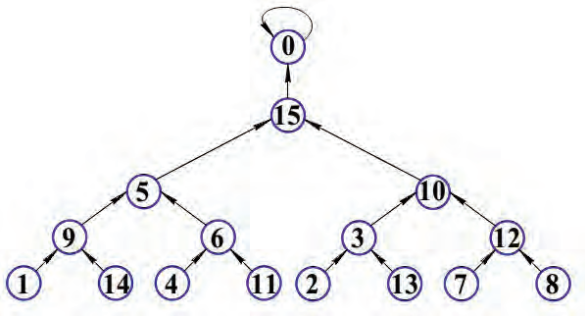
Разрядность 3



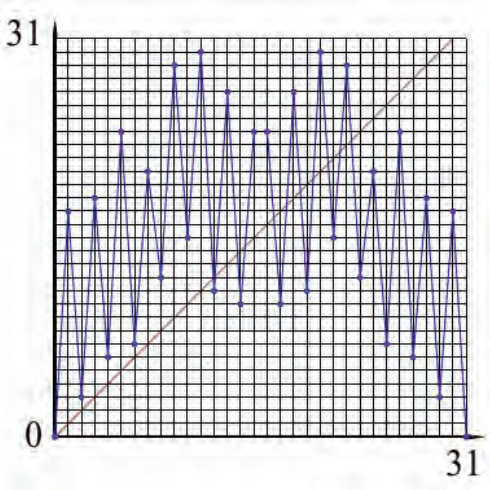


Разрядность 4





Разрядность 5



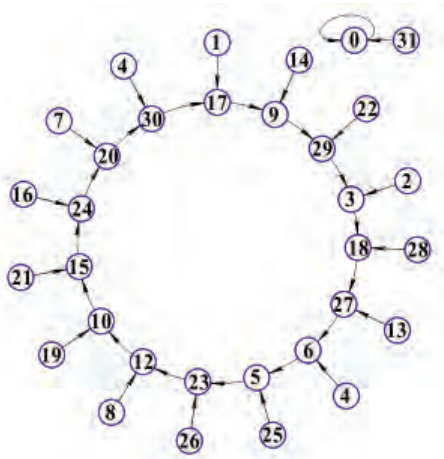


Рис. 12. Зависимость структуры графа монады от числа элементов образующего множества

Данные получены эмпирически и демонстрируют целый ряд нетривиальных закономерностей. Интересующий нас односвязный граф с топологией бинарного дерева проявляется при разрядности 4, затем наблюдается при разрядности 8 и далее повторяется с определённой периодичностью. На приведенных иллюстрациях мы вынуждены ограничиться простейшими картинками при малой разрядности регистров, но этого достаточно для понимания существенных изменений топологии графа монады при изменении числа исходных элементов. Этот эффект проявляется не для всех функций, образующих структуру. Отразить эти факты в аналитической записи структуры либо невозможно, либо очень непросто.

Мы примем за основу метод графового представления структур на конечных множествах с целью разработки дискретной динамики, как инструмента описания процессов в среде самоопределяемых данных.

4.2. Структуры на регистровых состояниях, граф кодовых переходов

Мы не будем рассматривать абстрактные множества, а ограничимся изучением вполне конкретных ситуаций во внутренней среде компьютера. Исходными множествами в нашем случае будут конечные наборы регистровых состояний, которые можно представлять, как битовые векторы определённой разрядности, либо записывать их как целые положительные натуральные числа. Операторы, образующие структуры на множествах регистровых состояний представляют собой дискретные преобразователи, которые можно реализовать аппаратно как логические схемы либо программно как наборы компьютерных команд. Оператор задаётся как функция и реализует функциональное отображение. Метод исследования структуры основан на представлении целочисленной функции в виде графа. Здесь требуется разъяснение – какая связь существует между функцией и графом.

Рассмотрим график целочисленной функции, заданной на конечном отрезке. Это будет совокупность точек на целочисленной решётке. Линия, соединяющая точки носит условный характер и может быть опущена. Совокупность точек на квадратной целочисленной решётке можно рассматривать как матрицу смежности, задающую ориентированный граф, рёбра которого есть совокупность пар вида ***x; F(x)***. Так, что представление функции в виде ориентированного графа в данной ситуации совершенно естественно, хотя и непривычно. Поскольку функция определена на регистровых состояниях, станем называть этот граф графом кодовых переходов и обозначать $\mathbf{G_F}$, граф, порождаемый функцией **F**.

Каждая точка графика целочисленной функции имеет две проекции - на ось ***x*** и на ось ***y***. Если график функциональный, то на все точки оси ***x*** всегда имеется одна и только одна проекция графика. На точки оси ***y*** может проецироваться любое число точек графика, в том числе и ни одной. Из этого следует, что на графе кодовых переходов каждая вершина всегда имеет одно и только одно исходящее ребро. Входных рёбер может быть сколько угодно от 0 до **N**, где **N** число состояний регистра. Следовательно, все возможные графы кодовых переходов образуют специфический класс, графов, ограниченный определёнными правилами структурообразования.

На рис. 13 можно проследить, как свойства графика функции F проецируются на свойства графа кодовых переходов

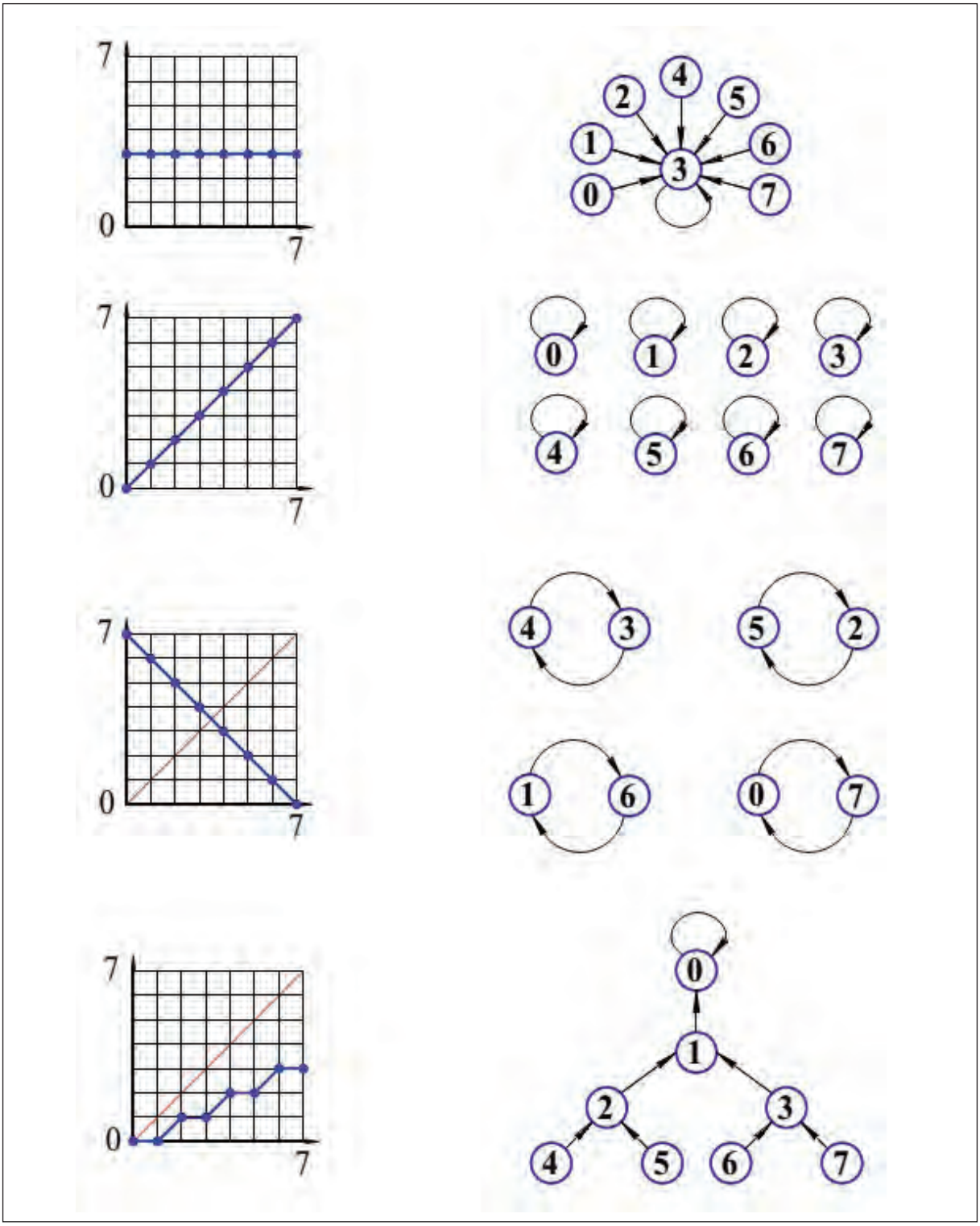


Рис. 13. Взаимосвязи графика функции и порождаемого графа $\mathbf{G}_F$

показано, что динамика поведения самоопределяемых данных порождается процессом смены значений теговых кодов, которые на каждом шаге обработки проходят через функциональный преобразователь. Когда значения функции используются в качестве аргументов, выполняется операция суперпозиции, и она применяется многократно, что изображено на рис. 14 а. А поскольку в этой длинной цепи на всех шагах преобразований функции **F** одна и та же, мы имеем операцию автосуперпозиции. Сокращенно операцию автосуперпозиции можно изобразить в виде замкнутой схемы, приведенной на рис. 14 б.

По сути это рекуррентный генератор, порождающий кодовые последовательности. Если мы хотим понять динамику развития кодовых последовательностей, порождаемых рекуррентным генератором мы должны обратиться к графу кодовых переходов. Порождаемые рекуррентным генератором кодовые последовательности есть маршруты на графе кодовых переходов и их динамика определяется структурой графа.

А теперь можно сделать определения основных понятий дискретной динамики:

Дискретный аттрактор это динамическая система, имеющая в своём составе множество регистровых состояний **R,** функциональный оператор **F,** отображающий **R** в **R** и оператор автосуперпозиции **F.**

Фазовый портрет дискретного аттрактора это граф кодовых переходов $\mathbf{G_F}$ порождаемый функцией **F.**

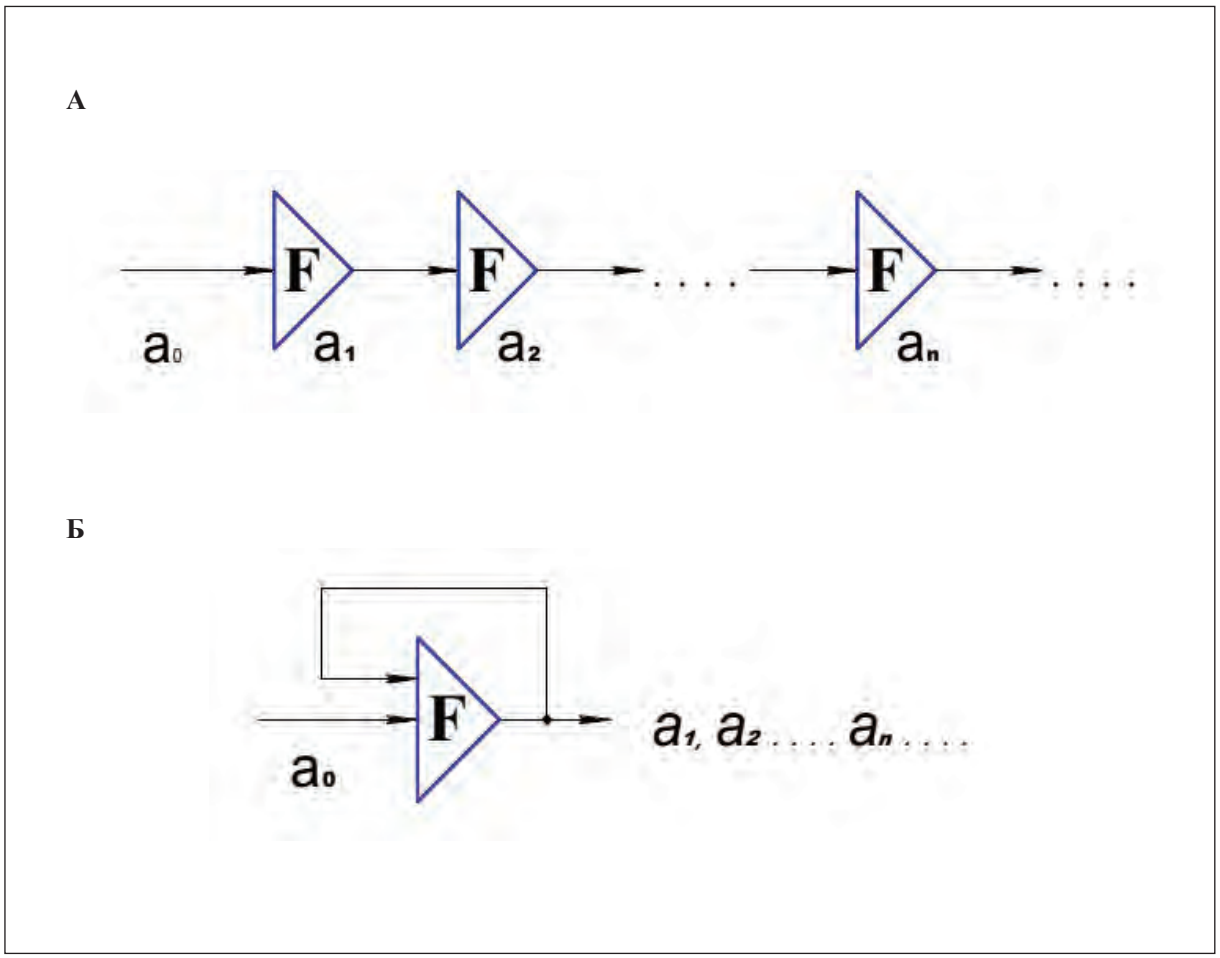


Рис. 14. Операция автосуперпозиции

Состояние совокупности самоопределяемых операндов в текущий момент фиксируется состояниями их теговых кодов. Теговые коды размещаются на вершинах графа кодовых переходов, т. е. на фазовом портрете дискретного аттрактора. Смена теговых кодов может осуществляться только переходом в смежные вершины на фазовом портрете.

Дискретный аттрактор задаёт функциональное пространство, в котором существуют самоопределяемые данные. Граф кодовых переходов формирует геометрию этого пространства как совокупность путей преобразования теговых кодов и именно таким образом исполняет роль фазового портрета аттрактора.

Силовой агрегат или движок, который осуществляет перемещения на фазовом портрете это процедура автосуперпозиции, реализованная как рекуррентный генератор. Рекуррентный генератор это процедура самообращения, подающая выходной результат вновь на вход преобразователя. Именно самообращающаяся процедура исполняет роль динамического ядра процесса.

Фазовый портрет это математическая абстракция, Граф кодовых переходов не реализуется физически, например, аппаратно как схемный лабиринт, по которому перемещаются самоопределяемые операнды или программно как сетевая структура данных. Фазовый портрет задаётся косвенным образом как результат взаимосвязанных трансформаций теговых кодов. Обширные фазовые портреты могут поддерживаться очень скромными аппаратными затратами. Если теговый код содержит 16 двоичных разрядов, а программа реализации оператора F состоит из менее чем десяти машинных команд, аттрактор поддерживает фазовый портрет в виде графа кодовых переходов, состоящего из сотен тысяч вершин. Уровень компрессии средств фиксации аттрактора в данном случае может составлять примерно 4 - 5 порядков. По этой причине появляется возможность не хранить детерминанты процесса в одном устройстве, осуществляющем сосредоточенное управление вычислениями, а разместить средства фиксации дискретного аттрактора непосредственно в каждом операнде и осуществить принцип распределённого управления.

Если все теговые сопровождения операндов обрабатываются одним преобразователем **F**, все они принадлежат одному фазовому портрету, а их теговые коды позиционируют всех и каждого на определённых вершинах графа кодовых переходов. В результате множество никак не связанных единым управлением операндов функционирует как строго согласованный ансамбль, двигающийся по заданным траекториям и реализующий заданный процесс. Уместно назвать этот эффект **функциональной когерентностью.**

Функциональная когерентность обеспечивается высокой степенью компрессии средств фиксации дискретного аттрактора. А компрессия возможна вследствие глубокого вырождения комбинаторики структурообразующих факторов, что означает потерю возможности задавать любые мыслимые и не мыслимые структуры фазовых портретов. В условиях компрессивного представления средств фиксации аттракторов возможные фазовые портреты образуют узкий класс структур, подчиняющихся жёстким ограничениям. Означает ли это потерю универсальности программирования любых приложений – вопрос дискуссионный. Во всяком случае, следует учитывать, что не любая последовательность команд классической машины может быть осмысленной программой.

Все кодовые траектории в дискретном аттракторе сходящиеся и завершаются попаданием в цикл на графе $\mathbf{G_F}$. Если цикл состоит из одной вершины, процесс останавливается. Это точка покоя дискретного аттрактора. Если цикл содержит множество вершин, аттрактор бесконечно повторяет заданную последовательность, что можно интерпретировать как постоянное циклическое выполнение определённого действия, например сканирования и контроля текущих параметров системы управления. Дискретный аттрактор обладает свойством устойчивости. Если внешнее воздействие или повреждение выбросит его из точки покоя, аттрактор сам, в соответствии с его устройством начнёт двигаться, а любые траектории движения аттрактора есть маршруты на фазовом портрете, которые всегда сходятся к точке покоя. В точке покоя механика аттрактора не выключается. Рекуррентный генератор кодовых последовательностей устроен таким образом, что в результате автосуперпозиции на выходе бесконечно порождается один и тот же код. Для внешнего наблюдателя это выглядит как останов. На самом деле дискретный аттрактор никогда не останавливается. Останов дискретного аттрактора носит условный характер, это стабильное состояние динамического равновесия, которое можно интерпретировать как непрерывный контроль равновесного состояния.

Дискретный аттрактор как динамическая система обладает определённой спецификой, которая в первую очередь заключается в том, что при формировании динамики не используется категория времени. Динамика дискретного аттрактора основана на отношениях предшествования, и каждое следующее состояние порождается как функция от предыдущего. В простейшем случае, когда параллельный процесс развивается в рамках одного аттрактора по единому фазовому портрету, все события увязаны отношениями предшествования, определяемыми графом кодовых переходов. Но при усложнении ситуации, предполагающей взаимодействие процессов, размещённых на разных аттракторах, придётся решать проблему определения одновременности событий в функциональных пространствах, в которых размещены и двигаются самоопределяемые данные. С этой целью придётся вводить понятия меры и измерения расстояния на фазовых портретах. При этом понадобится осуществлять переход от представления процесса как функции от предыдущего состояния к его представлению как функции от номера шага или длинны расстояния на фазовом портрете. Это одна из важнейших задач развития дискретной динамики.

4.4. Конкатенация

Актуальной задачей дискретной динамики является конструирование фазовых портретов с заданными свойствами или, что то же, построение графов кодовых переходов с заданной топологией. Первые шаги становления дискретной динамики сделаны чисто эмпирически. С помощью простой инструментальной программы задавались рекуррентные генераторы с определённой разрядностью регистров и определённой функцией преобразования кодов и для каждого генератора строился граф кодовых переходов $\mathbf{G_F}$. Таким образом, была сформирована базовая библиотека рекуррентных генераторов, представляющая базовые конструкции. Базовые конструкции известны и изображены на рис. 13. В вольной терминологии мы можем их перечислить. Это наборы самовозвратных полюсов, цепи, деревья, кольца, и розетки, представляющие собой деревья и цепи, пристыкованные к вершинам кольца. Самоовозвратными полюсами мы называем петли или минимальные циклы, состоящие из одной вершины. При этом граф может быть связен или иметь несколько компонент связности со своими циклами.

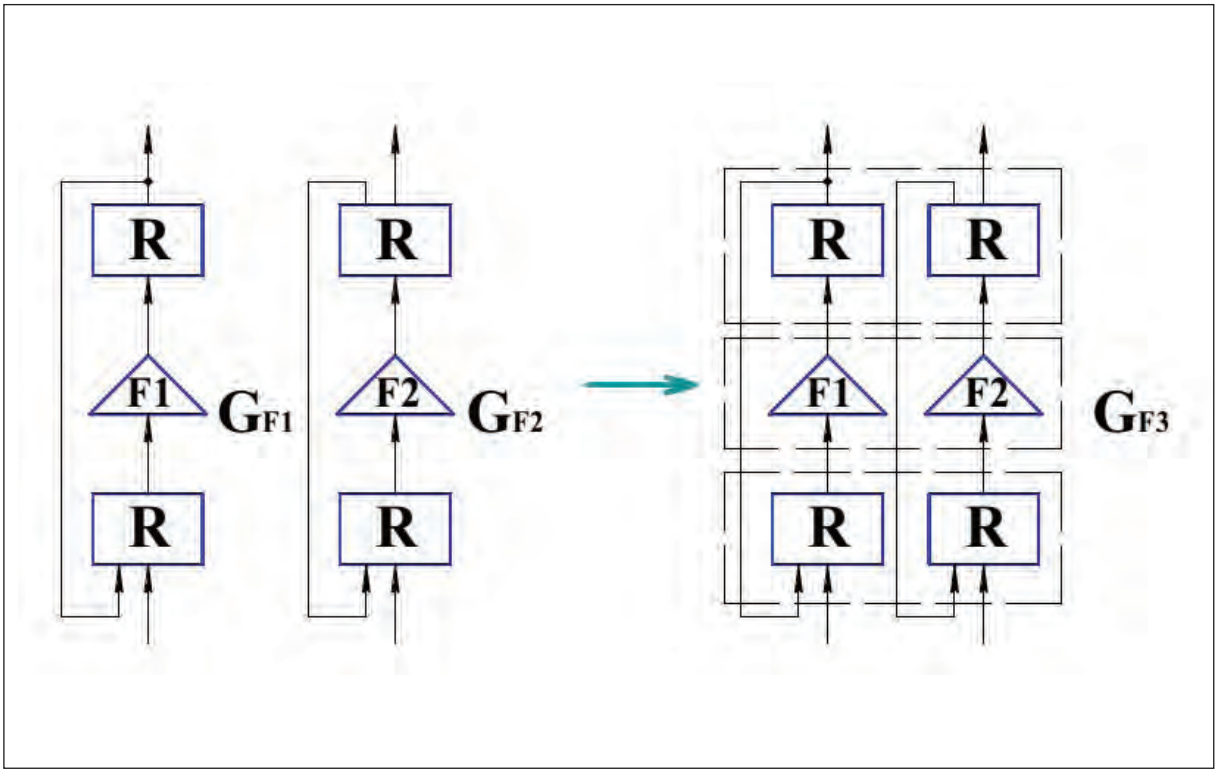


Рис. 15. Конкатенация рекуррентных генераторов

Для создания инструментальных средств дискретной динамики необходимо найти процедуру, позволяющую строить сложные конструкции фазовых портретов из простых базовых. Такая процедура существует и называется конкатенация. В наших условиях конкатенация означает совмещение двух и более регистровых секций в определённом порядке. В результате конкатенации образуется новый регистр, в котором правая секция выполняет роль младших разрядов, а левая секция роль старших разрядов объединённого регистра. Если регистры входят в состав рекуррентных генераторов и над ними построены функциональные преобразователи, в результате конкатенации образуется новый рекуррентный генератор, в котором и регистры и преобразователи образуются путём совмещения участников операции конкатенации. Иллюстрация операции конкатенации приведена на рис. 15.

Операция конкатенации объединяет два рекуррентных генератора, заданных функциями **F1** и **F2.** Каждый из участников операции порождает свой граф кодовых переходов $\mathbf{G_F1}$ и $\mathbf{G_F2}$. В результате конкатенации и совмещения двух исходных рекуррентных генераторов образуется новый результирующий, который порождает новый граф кодовых переходов $\mathbf{G_F3}$. Структура нового графа может существенным образом отличаться от структур участников конкатенации, а примитивная инженерная операция совмещения регистровых секций с математической точки зрения реализует нетривиальную операцию над графами. Далее приведены некоторые примеры операций над графами, возникающими при конкатенации рекуррентных генераторов.

Так, например, при конкатенации двух бинарных деревьев получается тернарное дерево. При этом максимальная глубина дерева сохраняется, а кратность ветвления умножается. Результаты конкатенации двух деревьев изображены на рис. 16.

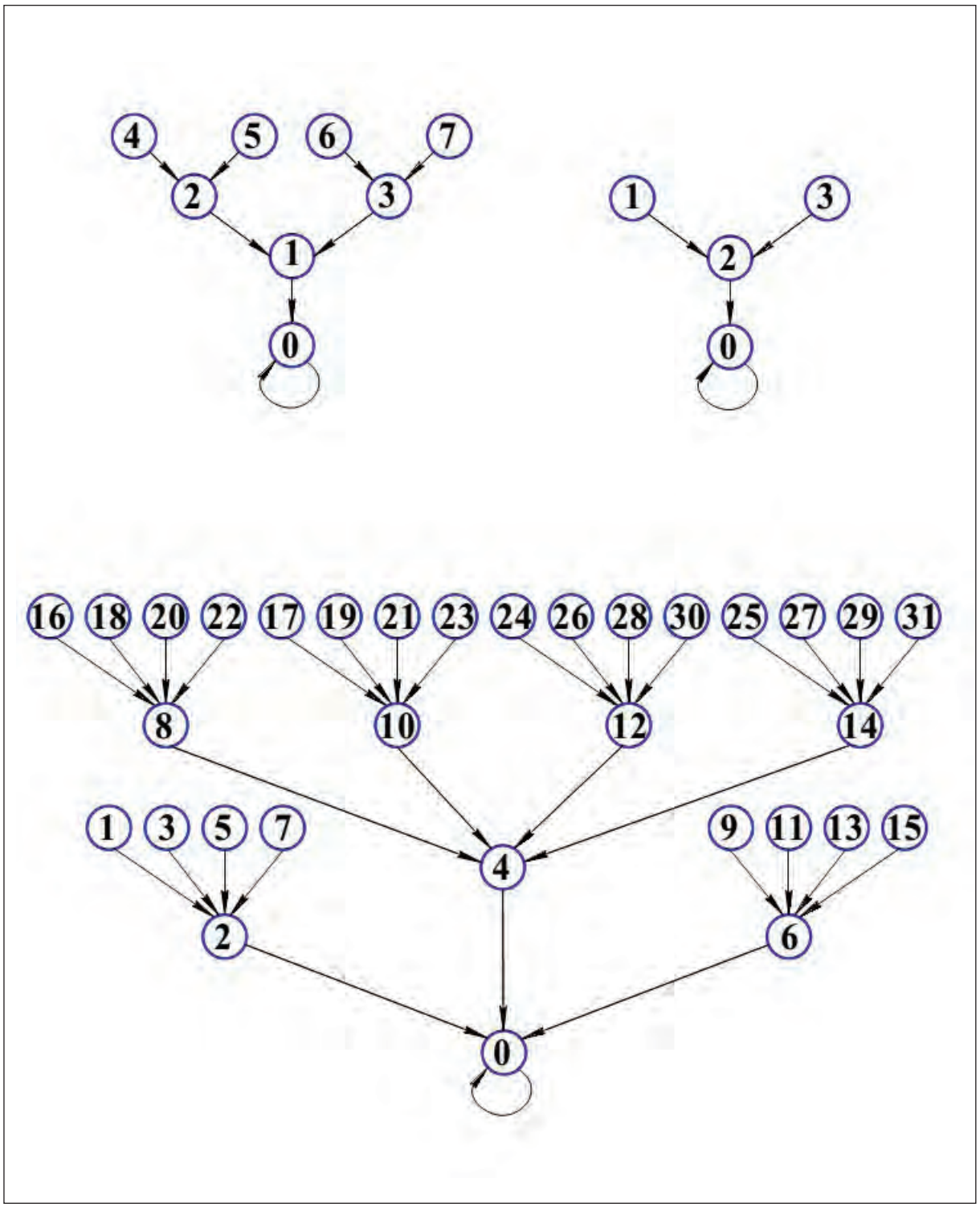


Рис. 16. Конкатенация двух деревьев

При конкатенации дерева и набора полюсов получается набор деревьев, т.е. связный граф превращается в многосвязный и число компонент связности равно числу полюсов в одном из участников конкатенации. Результаты приведены на рис. 17.

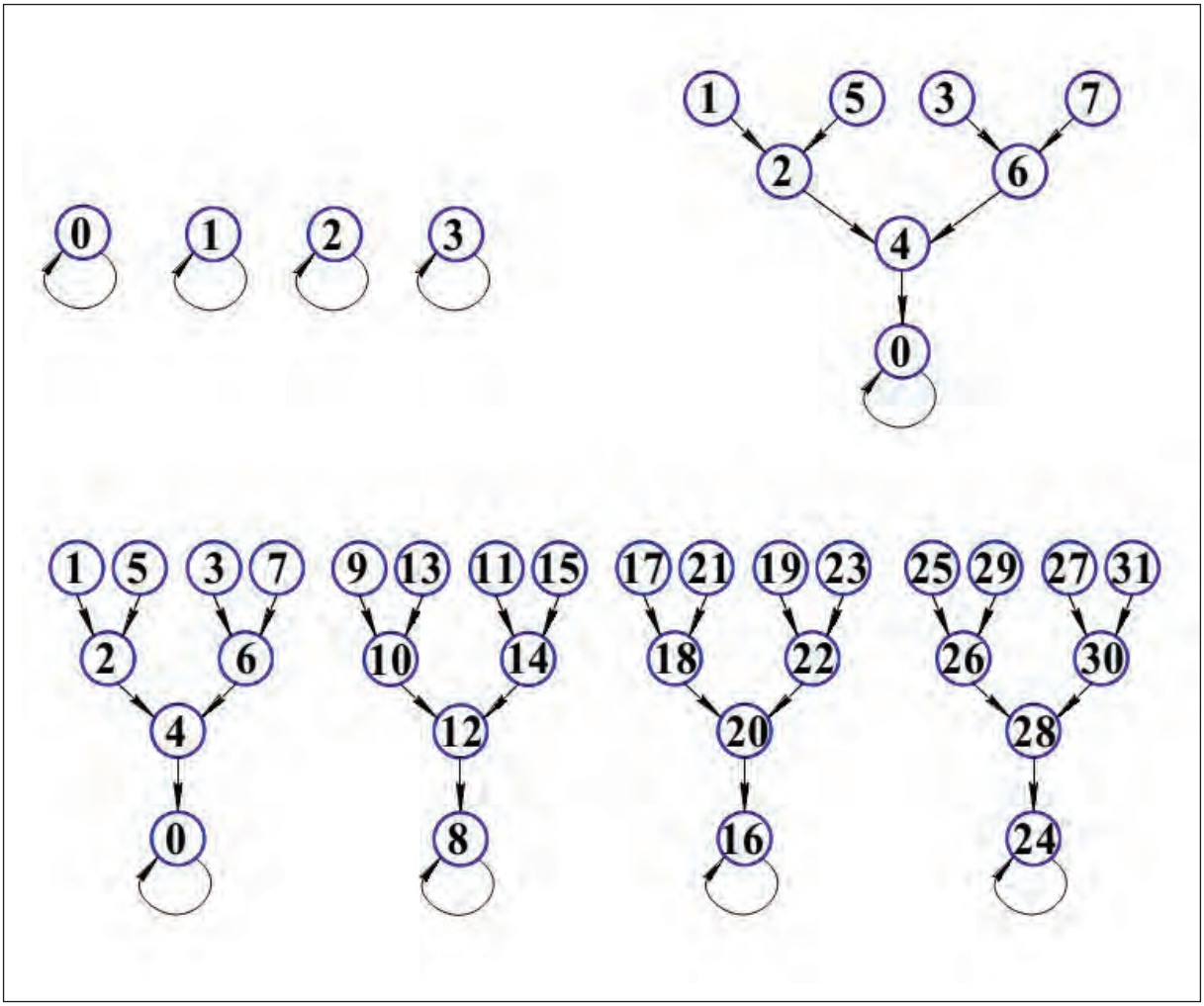


Рис. 17. Конкатенация дерева и набора полюсов

При конкатенации кольца и дерева, происходит образование розетки, представляющей собой набор деревьев пристыкованных к вершинам кольца.

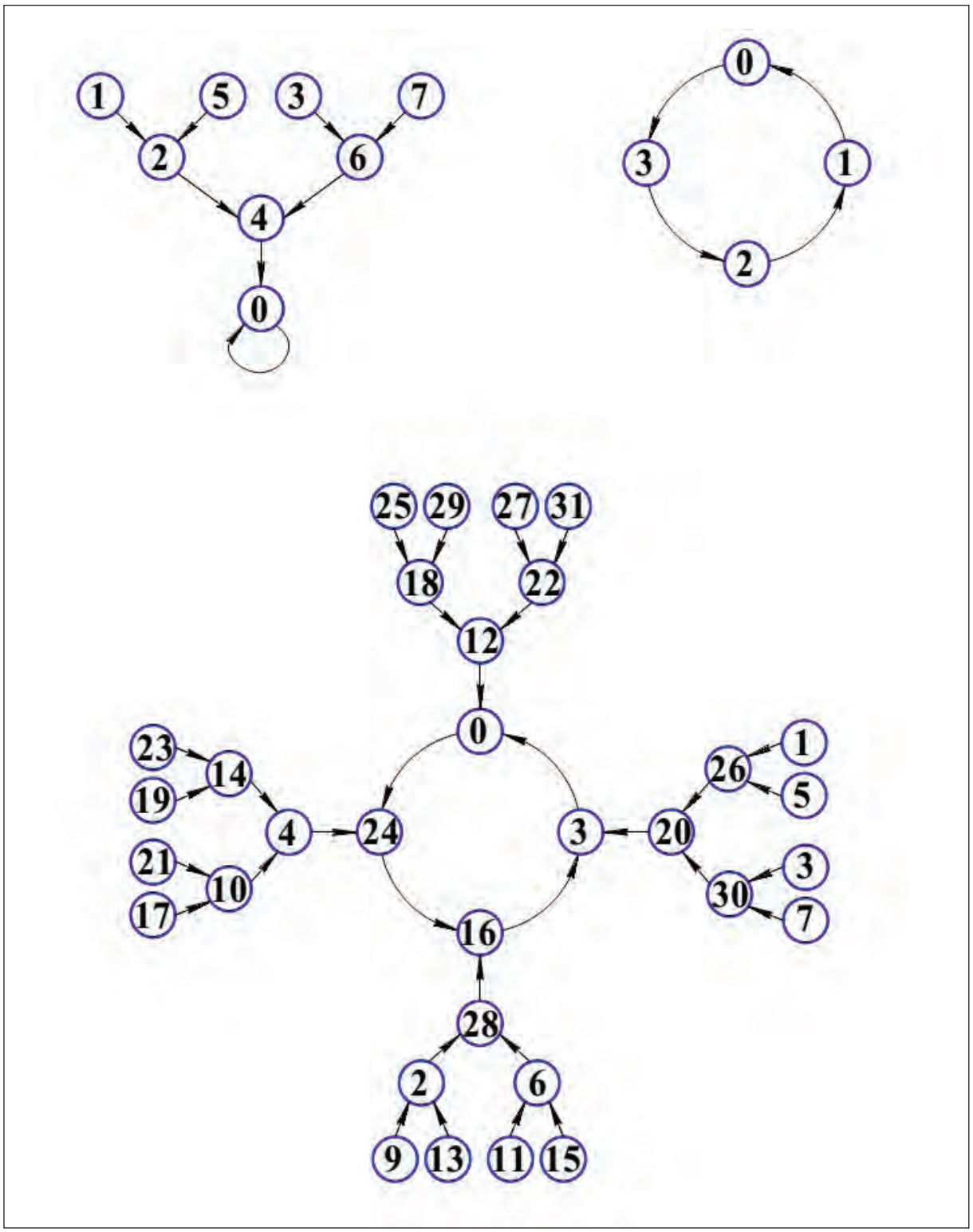


Рис. 18. Конкатенация дерева и кольца

Эмпирически удалось сформировать базу данных по конкатенации разных вариантов графов кодовых переходов и выделить набор практических приёмов конструирования структур, необходимых для решения ряда практических задач. Дальнейшая разработка этого направления должна привести к обоснованию свойств операций конкатенации и построению алгебры операций над графами кодовых переходов.

4.5. Репеллер и реверсивный аттрактор

До сих пор мы рассматривали варианты использования графа кодовых переходов как динамического ядра дискретного аттрактора, в котором продвижение по фазовому портрету происходит в направлении от листьев к корню дерева и циклу, которым завершается компонента связности. Для практических приложений дискретной динамики важно также обеспечить возможность продвижения по фазовому портрету в обратном направлении – от корня дерева к листьям. Эта задача не может решаться средствами функционального оператора, поскольку обратное отображение является многозначным и должно поддерживать ветвящийся процесс. Для поддержки ветвящегося процесса можно использовать гирлянду функциональных преобразователей. Гирлянда изображена на рис. 19.

Для примера выбрана гирлянда из трёх функциональных преобразователей, их число может меняться. Принцип действия гирлянды следует из её схемного изображения. При подаче на вход некоторого начального значения кода A_0 гирлянда откликается тремя выходными значениями A_1 , A_2 , A_3 . Далее каждое из них может подаваться на вход гирлянды и порождать на выходах следующие три значения. В данном случае рекуррентный генератор, построенный на базе гирлянды, порождает ветвящийся процесс, который носит расходящийся характер. Таким образом, вводится новый базовый элемент дискретной динамики - **дискретный репеллер. Дискретный репеллер** это расходящийся процесс, порождаемый гирляндой функциональных преобразователей и операцией автосуперпозиции гирлянды. Фазовый портрет репеллера это расходящийся граф кодовых переходов гирлянды. Репеллер по определению не имеет остановки и порождает бесконечный расходящийся процесс. Вследствие конечности набора кодов, на котором определены функциональные преобразователи гирлянды рано или поздно запас кодов исчерпывается и на выходе гирлянды появляется код, который ранее уже имел вхождение в вершины графа кодовых переходов. Это означает, что фрагмент дерева, начинающийся с данной вершины, будет воспроизведен репеллером вновь.

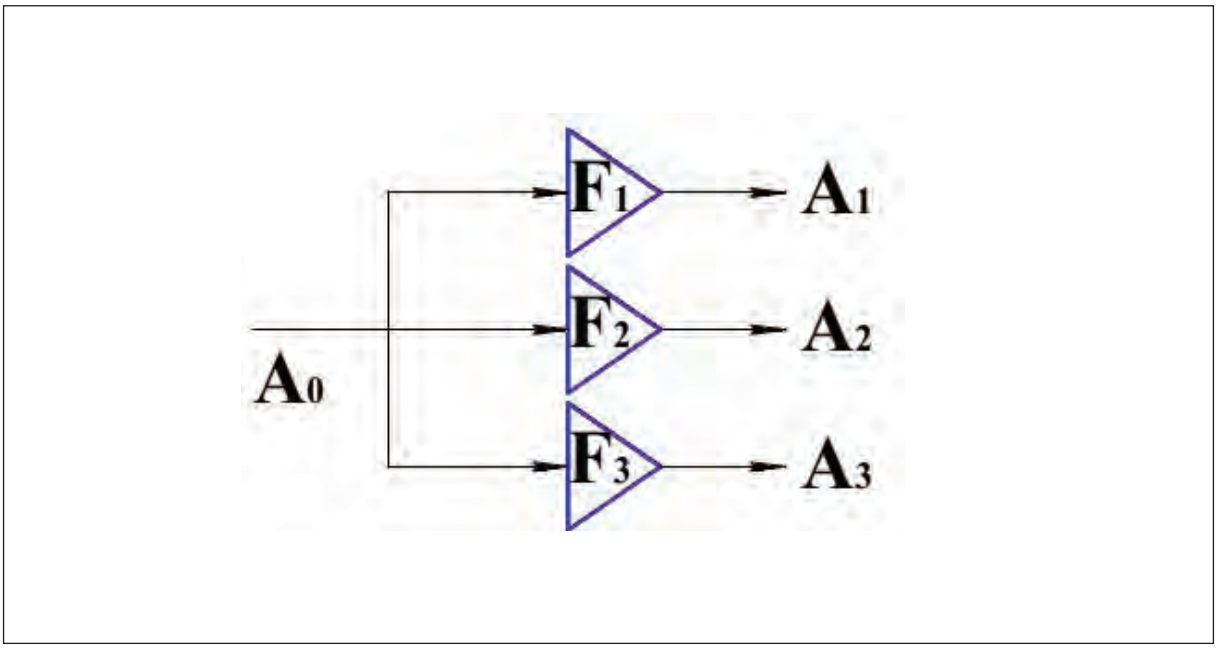


Рис. 19. Гирлянда, образующая дискретный репеллер

В целом фазовый портрет репеллера будет состоять из бесконечного повторения конечного набора фрагментов, графа кодовых переходов, заданного гирляндой. Исследование динамики развития процессов в репеллерах это следующий и очень содержательный раздел дискретной динамики.

Опираясь на понятия дискретный аттрактор и дискретный репеллер можно сформулировать важное для приложений дискретной динамики понятие **реверсивный аттрактор.** Допустим, что задан дискретный аттрактор с определенным фазовым портретом. Для построения реверсивного аттрактора необходимо синтезировать репеллер, фазовый портрет которого совпадает с фазовым портретом аттрактора с точностью до инверсии направления рёбер графа кодовых переходов. Реверсивный аттрактор это сопряжённая пара аттрактора и репеллера, обеспечивающая продвижение процесса по заданному фазовому портрету в прямом и обратном направлениях. Задача эта решается следующим образом. Исходный аттрактор задаётся рекуррентным генератором на базе определённой целочисленной функции. График функции представляется как совокупность точек на целочисленной квадратной решётке. График изображён на рис. 20 а.

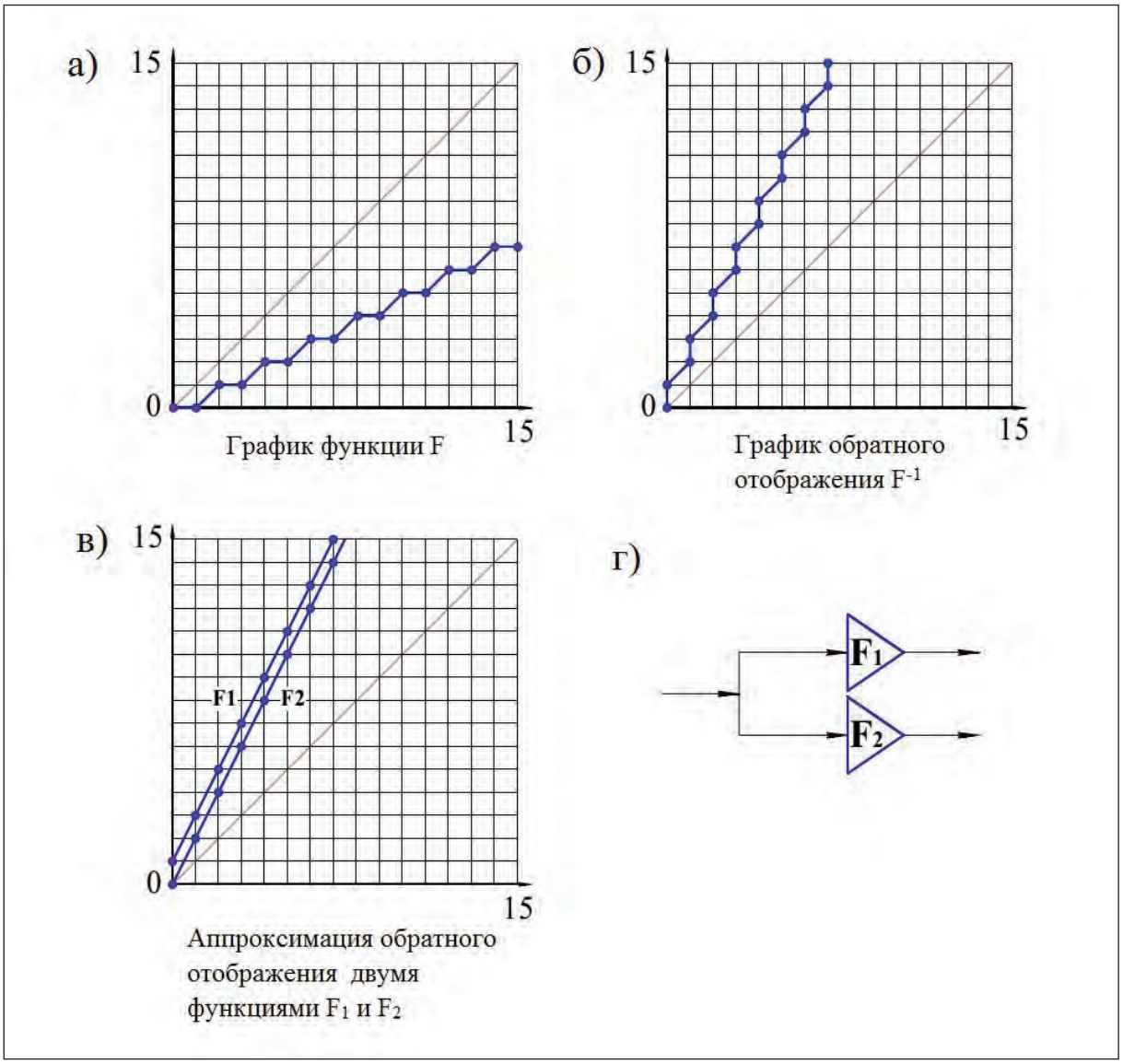


График функции F

График обратного отображения F^{-1}

Аппроксимация обратного отображения двумя функциями F_1 и F_2

Рис. 20. Операции построения семейства сопряженных функций для реверсивного аттрактора

По данному графику необходимо построить обратное отображение. Для этого надо выполнить операцию замены координат, что достигается поворотом исходного графика вокруг главной диагонали. График обратного отображения изображён на рис. 20 б. Обратное отображение не является функциональным и далее надо решить задачу синтеза набора сопряжённых функций, покрывающих, полученный график обратного отображения. Перечисленные действия изображены на рис. 20.

Эта задача не имеет общего решения. В каждом конкретном случае путём подбора можно найти множество вариантов покрытия графика обратного отображения функциональными графиками и искать среди них приемлемые по критерию удобства реализации сопряжённых функций. Из полученного таким образом набора сопряжённых функций составляется гирлянда для реализации репеллера, дополнительного к заданному аттрактору.

4.6. Маскирование и сильное ветвление

Рассмотренные до сих пор средства формирования графов кодовых переходов предполагают, что каждое регистровое состояние из множества, образующего структуру имеет однократное вхождение в граф кодовых переходов. В практике применения дискретных аттракторов для формирования вычислительных работ необходимо обеспечить многократную повторяемость кодов операций на графе вычислительного процесса. Это достигается путём наложения маски на образующие регистры и чтения из под маски только части разрядов, необходимых для кодирования операций. Рассмотрим конкретный пример. Необходимо построить граф кодовых переходов со структурой сильно ветвящегося дерева с кратностью ветвления равной 8 и заданной глубиной 4 уровня. При этом ставится задача обеспечить возможность полной нумерации предшественников каждой вершины. Выберем определённую вершину и рассмотрим набор из восьми вершин, являющихся предшественниками данной. Коды вершин предшественников должны быть устроены таким образом, чтобы при наложении на них определённой маски можно было извлечь три разряда, задающих полную нумерацию предшественников от 0 до 7, или в двоичном выражении от комбинации 000 до 111. По принятому условию это свойство должно выполняться для всех вершин кроме листовых, которые не имеют предшественников.

Искомый рекуррентный генератор, поддерживающий заданное сильно ветвящееся дерево будет сформирован в результате конкатенации трёх генераторов, поддерживающих четырёхэтажные бинарные деревья и построенные на базе уже известной нам функции сдвига вправо на один разряд. По известным нам свойствам операции конкатенации бинарных деревьев глубина сохраняется равной 4, а кратности ветвления перемножаются и дают требуемое значение 8. В итоге получается полное четырёхэтажное дерево с кратностью ветвления равной 8. А теперь наложим на образующие регистры маску, которая позволяет извлечь из каждой секции участвующей в конкатенации один младший крайний правый разряд. Трехразрядный код, извлечённый в результате маскирования, содержит полную нумерацию вершин предшественников. Остальные разряды полного кода позиционируют вершины на дереве в целом. На рис. 21 приведен фрагмент сильно ветвящегося дерева с реальными значениями кодов вершин и показано извлечение полной нумерации вершин предшественников.

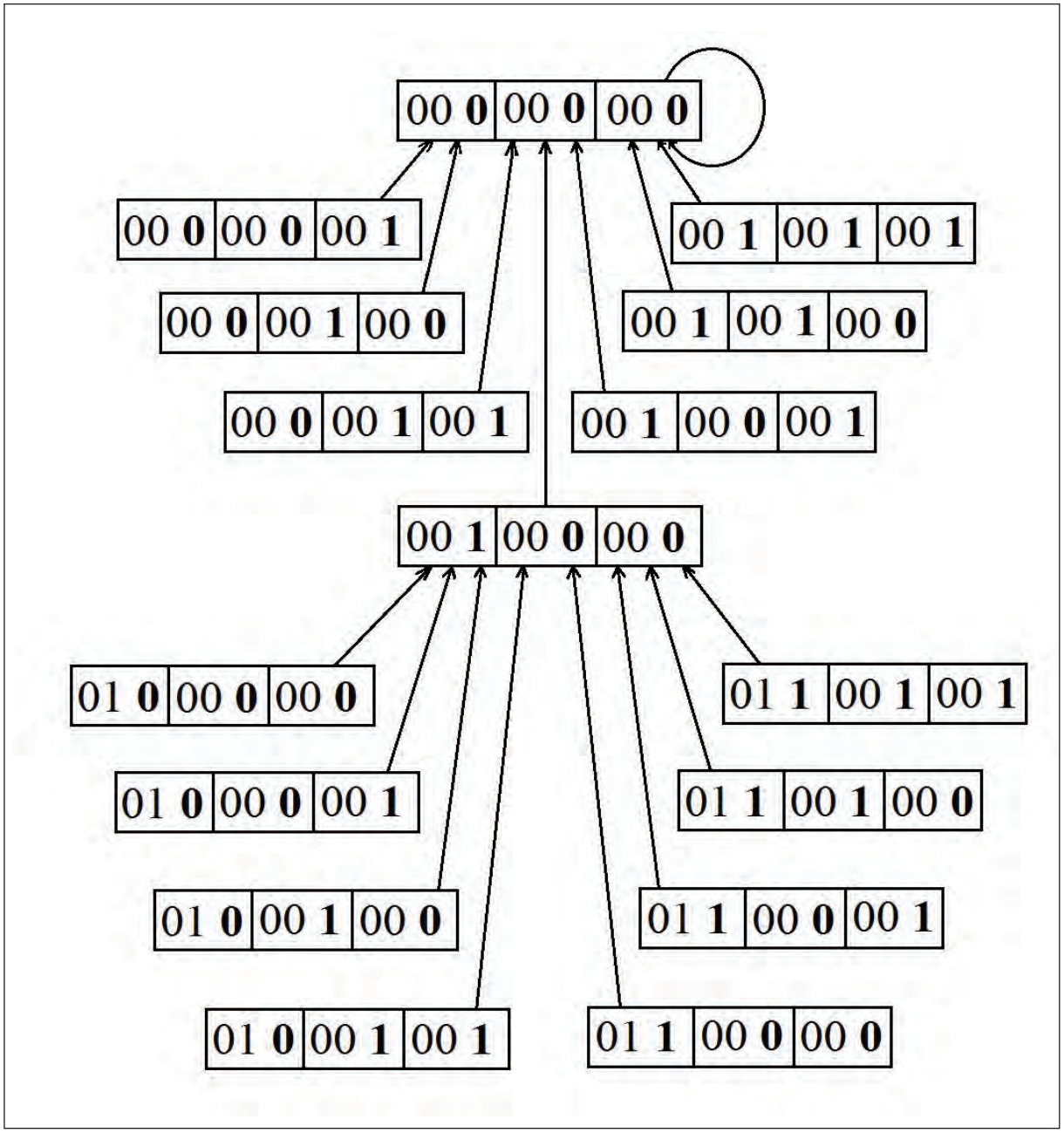


Рис. 21. Фрагмент сильно ветвящегося дерева и результаты маскирования кодов вершин предшественников

Разряды кодов, извлекаемые маской, обозначены жирным шрифтом и отделены пробелом.

4.7. Компилятор и технология программирования самоопределяемых данных

Весь объём изложенных выше понятий и механизмов дискретной динамики необходим для обсуждения возможной технологии программирования вычислений в архитектуре самоопределяемых данных. Задача ставится следующим образом: задано арифметическое выражение, по которому строится граф вычисления в виде бинарного дерева и затем синтезируется дискретный аттрактор, фазовый портрет которого накрывает этот граф. Для решения поставленной задачи построим дискретный аттрактор с фазо-

вым портретом в виде сильно ветвящегося дерева, в котором содержатся бинарные поддеревья вычисления всех возможных арифметических выражений, не превышающих определённый уровень сложности, ограниченный глубиной дерева вычисления. Работа аттрактора запускается загрузкой в вычислитель набора исходных операндов. Теговые коды исходных операндов позиционируют начальное состояние процесса на фазовом портрете. Дальнейшие события разворачиваются детерминировано и определяются конструкцией аттрактора и структурой фазового портрета. Правильно загруженный набор исходных операндов извлекает из фазового портрета своё бинарное поддерево. Следовательно, для программирования заданного вычисления необходимо отыскать, соответствующие ему листовые вершины и присвоить их коды исходным операндам в виде теговых сопровождений.

Для отыскания листовых вершин заданного вычисления необходимо наложить заданное бинарное дерево на общее сильно ветвящееся. Предполагается, что нумерация вершин предшественников увязана в таблицу кодирования арифметических операций. Размещение начинается с корневой вершины. Выбирается заключительная операция заданного вычисления и ставится маркер на вершину предшественницу корня, номер которой соответствует заключительной операции. Два операнда, участвующие в заключительной операции порождаются выполнением двух предшествующих арифметических операций, что обнаруживается при раскрытии скобок в направлении от главной заключительной операции. На следующем шаге необходимо поставить маркерные метки на двух вершинах, предшествующих отмаркированной на первом шаге. При этом следует отбирать вершины с номером, соответствующим кодам предшествующих арифметических операций. Если бы предшествующие операции всегда были разными, можно было бы ограничиться ветвлением общего дерева с кратностью равной четырём. Но в общем случае обе предшествующие операции могут быть одинаковыми. По этой причине в номерах предшествующих вершин каждая операция должна быть представлена дважды. Вот почему при четырёх операциях понадобилось дерево с кратностью ветвления равной восьми. Описанная процедура повторяется до исчерпания операций в исходной записи арифметического выражения. Последние маркерные метки отмечают листовые вершины бинарного дерева искомого вычислительного процесса. Результат программирования в данном случае это назначение теговых кодов исходным операндам. При загрузке в вычислитель исходные операнды запускают вычислительный процесс и извлекают требуемое бинарное поддерево путём прохождения своих маршрутов на фазовом портрете.

Разработанные ранее инструментальные средства дискретной динамики позволяют сконструировать компилятор, осуществляющий программирование процессов вычисления арифметических выражений в архитектуре самоопределяемых данных. Для этого есть инструменты формирования сильно ветвящихся деревьев, есть инструменты маскирования кодов для организации полной нумерации вершин предшественников, есть методы построения реверсивных аттракторов, обеспечивающих движение по фазовому портрету в прямом и обратном направлениях. Описанный компилятор можно сделать полностью автоматическим.

Поддержка полного сильно ветвящегося дерева необходима для работы компилятора, который должен содержать пути для вычисления всех возможных арифметических выражений. После выбора заданного выражения и завершения компиляции поддержка полных теговых кодов при исполнении процесса становится избыточной. Возможна оптимизация, которая означает сокращение разрядности тегов. Сокращение раз-

рядности теговых кодов должно происходить при условии, что общая конструкция графа кодовых переходов не нарушается, выбранное бинарное поддерево сохраняется, а лишние ветви полного сильно ветвящегося дерева отсекаются. Такая задача может быть поставлена и есть предположения по её решению. Задача не имеет общего решения, это означает, что в каждом конкретном случае эффективность сокращения разрядности теговых сопровождений будет различной.

Заключение

Достигнутый уровень технологии микроэлектронного производства обусловил основные направления развития вычислительных средств на ближайшее десятилетие. В соответствие с законом Мура отрасль будет рентабельна при условии соблюдения определённых темпов роста производительности вычислительных средств и выполнения определённых технологических ограничений, в частности соблюдением заданных показателей энергоэффективности. В целом это означает, что главным инструментом роста производительности становится освоение массового параллелизма вычислений. В данной работе проведено исследование динамики роста производительности высокопараллельных структур и выявлен целый ряд парадоксальных явлений. На примере конкретной, но типичной задачи показано, что максимум производительности достигается в диапазоне нескольких сотен процессорных элементов, а в области высоких значений параллелизма (десятков и сотен тысяч) производительность катастрофически падает. В диапазоне пика производительности эффективность использования потенциала параллелизма не превышает 10 %. Определяющим фактором при этом является архитектурная идея. Принятые формы организации вычислений базируется на разделении зоны хранения и зоны обработки данных, которые строятся по различным технологиям. Анализ баланса временных затрат показывает, что львиная доля приходится на пересылки данных из зон хранения в зоны обработки и обратно. Это определяет показатель эффективности процессорного элемента, который не превышает 1%. Выражаясь метафорически на одно полезное действие, собственно обработки данных, приходится не менее 100 действий, обеспечивающих доступ в память и пересылки данных. Попытка увеличения производительности вычислений за счёт параллелизма и использования множества процессорных элементов требует разрезки информационной структуры алгоритма и размещения её в разных процессорных элементах. А это приводит к тому, что часть пересылок обретает статус межпроцессорных обменов и требует особой коммутационной среды. По мере увеличения параллелизма информационная структура алгоритма дробится на всё более мелкие фрагменты, и всё более и более крупные объёмы пересылок становятся межпроцессорными. Для больших значений параллелизма баланс временных затрат обретает гипертрофированный характер. Доля затрат на вычисления становится исчезающе малой, а доля обменных операций становится быстро растущей доминантой и никакие усложнения и ускорения коммутационной среды не могут переломить эту динамику. Увеличение степени параллелизма при больших значениях приводит к устойчивому падению производительности.

Далее есть два пути - сохранение классической архитектуры и поиск задач большой размерности с малой связностью данных и отказ от классической архитектуры и разработка новых форм организации вычислений и новых технологий программирования. Каждое из этих двух направлений займёт свою нишу в широком спектре применений. В данной работе излагается пример разработки принципиально новой архитектурной

идеи, основанным положением которой является совмещение зон хранения и обработки данных. Архитектура самоопределяемых данных поддерживает массовый мелкозернистый параллелизм на уровне обработки отдельных операндов и при этом не порождает крупные массивы межпроцессорных обменов. Принципы, заложенные в архитектуре самоопределяемых данных, создают предпосылки для создания новых технологий программирования, основанных на применении специальных математических инструментов. В статье даётся краткий очерк основ дискретной динамики и приводится пример использования основных понятий дискретной динамики для моделирования и программирования вычислений в системе самоопределяемых данных.

Литература

1. Сборник трудов 7-й Международной конференции по физике и технологии гетероструктурной СВЧ-электроники 25 мая 2016 г. // Москва МИФИ, 2016, 169 стр.
2. *Duato J., Yalamanchili S., Ni L. M.* Interconnection networks: an engineering approach. // Morgan Kaufmann, 2003, 600 pp.
3. *Стегайлов В. В., Норман Г. Э.* Проблемы развития суперкомпьютерной отрасли в России: взгляд пользователя высокопроизводительных систем // Программные системы: теория и приложения, 2014, 1 (1), 111-152.
4. *Норман Г. Э. и др*. Зачем и какие суперкомпьютеры экзафлопсного класса нужны в естественных науках // Программные системы: теория и приложения, 2015, 6 (4), 243-311.
5. *Kim J., Dally W. J., Abts D.* Flattened butterfly: a cost-efficient topology for high-radix networks // ACM SIGARCH Computer Architecture News, 2007, 35 (2), 126-137.
6. *Yadav S., Mishra R., Gupta U.* Performance evaluation of different versions of 2D Torus network // Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in IEEE, 2015, 178-182.
7. Resources for Embedded and IoT Designers // URL: http://www.intel.com/design/network/products/optical/cables/index.htm.
8. *Dally W. J., Poulton J. W.* Digital systems engineering // Cambridge University Press, New York, NY, 1998, 663 pp.
9. *Махиборода А.В., Ильичёв А.В., Подобин А.А*. Проблемы реализации массового динамического параллелизма I // Наноструктуры математическая физика и моделирование, 2016, 14 (1), 47-67.
10. *Dennis J. B.* First version of a data flow procedure language // Programming Symposium, Springer Berlin Heidelberg, 1974, 362-376.
11. *Дурнев В. Г., Стандрик В. Д.* Основы построения систем передачи ЕАСС. // М.: Радио и связь, 1985, 208 стр.
12. *Мизин И. А., Богатырев В. А., Кулешов А. П*. Сети коммутации пакетов. //М.: Радио и связь, 1986. 408 стр.
13. *Махиборода А.В. и др.* Система потоковой обработки информации с интерпретацией функциональных языков // Авторское свидетельство № 1697084 опубликовано 29. 06. 1989 г. Положительное решение 25.05 1990 г.
14. *Мизин И. А., Махиборода А. В*. Концепция самоопределяемых данных и архитектура распределённых систем // Информационные технологии и вычислительные системы, Москва, Наука, 1995, 1, 21-31.
15. *Махиборода А. В*. Проблемы создания вычислительных средств с непроцедурными внутренними языками // Управляющие системы и машины, 1993, 3, 51-62.
16. *Арнольд В. И.* Топология и статистика формул арифметики // Успехи математических наук, 2003, 58, 4 (352), 3-28.
17. *Арнольд В. И.* Топология алгебры: комбинаторика операции возведения в квадрат // Функциональный анализ и его приложения, 2003, 37 (3), 20-35.
18. *Арнольд В. И.* Сложность конечных последовательностей нулей и единиц и геометрия конечных функциональных пространств //Публичная лекция, 2006, 13 стр.

PROBLEMS OF MASSIVE DYNAMIC PARALLELISM IMPLEMENTATION. II

A.V. Makhiboroda, A.V. Ilichev, A.A. Podobin, A.V. Tsarev

Department of Applied Mathematics MIEM,
National Research University HSE

makhiboroda@yandex.ru

Received 17.05.2016

In the first part of this paper, it is noticed that the development of massive dynamic parallel computations becomes the main tool for ensuring the stable growth of the processing power of computing facilities. It is also shown that the implementation of massive parallel computations meets two major obstacles. These are specific costs of the parallelism support which grow faster than the number of processors and a poor performance of the interconnection networks supporting the interprocessor communications.

In the second part of this paper, the forecasts of the interconnection network development are evaluated from the standpoint of improving its algorithms and increasing its bandwidth. The expert opinions about the possibilities of extending the effective parallelism range to the value of hundreds of processors are given. At the same time, systems with several thousands of processors on a chip have already been constructed, and the current state of the art in these technologies allows predicting an increase in the number of processors to more than tens of thousands in the next few years. Searching the ways to extend the effective implementation range of parallelism is a rather urgent problem directed to develop principally new forms of algorithms. The basic ideas of self-defined data architecture are discussed and a brief outline of the construction of discrete dynamics mathematical formalism is given. The conceptual apparatus of discrete dynamics allows formulating and solving the problem of programming the processes in self-defined data architecture.