УДК 681.324-192

ПЕРСПЕКТИВЫ РАЗВИТИЯ ЦИФРОВЫХ СИГНАЛЬНЫХ ПРОЦЕССОРОВ И ВОЗМОЖНАЯ РЕАЛИЗАЦИЯ РЕКУРРЕНТНОГО ОБРАБОТЧИКА СИГНАЛОВ

Степченков Ю.А., Петрухин В.С.

Архитектура рекуррентного обработчика сигналов (РОС) изначально разрабатывается как специализированная, предназначаемая для реализации параллельных вычислительных процессов (ВП) обработки сигналов в реальном времени. Она базируется на рекуррентно-динамической вычислительной парадигме, которую можно рассматривать как развитие парадигмы потока данных (Data flow-парадигмы), но построенной на другом принципе, а именно - на принципе самодостаточных, рекуррентно сжатых данных (РД). Как и в других потоковых архитектурах (ПА), формат РД в РОС подразумевает наличие функциональных полей (ФП). Самодостаточность РД проявляется двояко. Во-первых, ФП содержат всю информацию по их обработке (в отличие от традиционных ПА, содержащих лишь ссылку на адрес команды, которая должна быть привлечена для обработки на текущем шаге ВП). Во-вторых, ФП содержат информацию по обработке данных не только на текущем шаге, но на ряде шагов, в идеале - до завершающего шага обработки; таким образом, РД способны нести в себе рекуррентно свёрнутый алгоритм решения задачи.

К настоящему моменту подтверждена эффективность разрабатываемой рекуррентной архитектуры только на модели ее операционного уровня - многосекционного рекуррентного операционного устройства (РОУ) [1]. При этом степень детализации модели соответствует концептуальному уровню проработки РОУ. Рассматривались и анализировались только алгоритмические аспекты реализации ограниченного набора алгоритмов над целочисленными данными с использованием четырех элементарных команд Вычислителя. Вычислительные аспекты, связанные с точностью представления входных данных и получаемых результатов, вопросы округления, переполнения и т.д. не рассматривались. Таким образом, подтвержденная эффективность РОС носит предварительный и ограниченный характер. В настоящее время разрабатывается детализированная архитектура РОУ, в связи с чем возникает необходимость проанализировать современное состояние, тенденции и перспективы развития цифровых сигнальных процессоров (ЦСП). С другой стороны, нужно выявить факторы, препятствующие ши-

рокому использованию потенциальных преимуществ ПА в организации параллельных систем, в том числе в ЦСП.

1. Тенденции развития цифровых сигнальных процессоров

Проиллюстрировать исторический путь развития архитектур ЦСП можно сменой их поколений. Понятие "поколение" не имеет однозначного толкования. Одни фирмы-производители ЦСП считают каждое свое новое семейство процессоров началом выпуска нового поколения процессоров, т.е. отождествляют понятия "семейство" и "поколение". Другие фирмы, например, Texas Instruments (TI), в качестве классификационного понятия используют термин "платформа", в рамках которой они выделяют отдельные семейства, в свою очередь, подразделяющиеся на поколения (понятие нижнего уровня).

Существует и другая точка зрения, которая в качестве классификационного понятия выделяет наиболее существенные (определяющие их характеристики и, соответственно, области применения) черты ЦСП, претендующие на особый статус их архитектуры. В табл. 1 отражена эволюция развития ЦСП, представляющая собой расширенный и детализированный вариант "скелета" такого представления из [2]. Данные этой таблицы свидетельствуют о том, что законодателем мод в области ЦСП является фирма TI - в семи поколениях из девяти представлены ее изделия.

Эволюция архитектур ЦСП претерпела радикальные изменения - от простых, относительно низкопроизводительных однопроцессорных ЦСП фон-неймановской архитектуры до высокопроизводительных многопроцессорных однокристальных суперскалярных масштабируемых ЦСП модернизированной VLIW-архитектуры.

Эволюция архитектур ЦСП в значительной степени воспринимает новации процессоров общего назначения. Справедливо и обратное утверждение. Учитывая важность такого заключения (*взаимопроникновения* архитектурных принципов, методов и приемов), в том числе и для прогнозирования тенденций развития архитектур ЦСП в ближайшей перспективе, остановимся на этом более подробно.

1) Известно, что большой класс математических функций, требующих для своей реализации больших вычислительных ресурсов, может быть сведен к серии умножений и сложений. Аппаратура ряда микропроцессоров, базирующихся на фон-неймановской парадигме, была оптимизирована для быстрого выполнения умножения.

Таблица 1. Эволюция ЦСП-процессоров

Покол.	Характеристики	Примеры
0 (1980)	Фон-неймановская архитектура	DSP-1 (AT&T)
1 (1982)	Модернизированная гарвардская архитектура (передача данных из памяти программ в память данных, двухступенчатый конвейер команд). ШП, ШД и 2 соответствующие АШ	TMS32010 (TI), NEC7720
2 (1986)	Модернизированная гарвардская архитектура (передача данных из памяти программ в память данных, 4-ступенчатый конвейер команд). ШП, ШДЧ, ШДЗ и 3 соответствующие АШ	TMS320C2x (TI), DSP16A (AT&T)
3 (1989)	Комбинированная гарвардская/фон-неймановская архитектура (общее адресное пространство программ, данных и ввода/вывода, кэш команд)ШП, ШД, ШД_ПДП; 3 АШ и дополнительная АШ данных	TMS320C30 (TI)
4 (1990)	Дополнительные режимы адресации, дополнительные функции	TMS320C5x (TI), DSP16xx (AT&T)
5 (1991)	Комбинированная гарвардская/фон-неймановская архитектура с элементами суперскалярности (до 8 параллельных операций в одном ПЭ; раздельные глобальные шины адреса/данных для многопроцессорных конфигураций, до 6 высокоскоростных портов). Организация внутренних шин и кэш соответствует TMS320C30	TMS320C4x (TI)
6 (1994)	Усовершенствованная модернизированная гарвардская архитектура (набор функциональных устройств: декодер экспоненты, акселератор Витерби, устройство сравнения и селекции; набор команд для их параллельной работы и эффективной обработки комплексных чисел).ШП, 3 ШД и 4 соответствующие АШ	TMS320C54x (TI)
6* (1994)	Усовершенствованная модернизированная гарвардская архитектура (два 17-разрядных MAC-устройства, четыре 40-разрядных аккумулятора) ШП, 3 ШДЧ, 2 ШДЗ, ШДП, ШД_ПДП и 8 соответствующих АШ	TMS320C55x (TI)
7 (1994)	Однокристальная многопроцессорная суперскалярная масштабируемая архитектура, четырехступенчатый конвейер команд (параллельность на уровне задач - до 4 ADSP (Advanced DSP) на кристалле; параллельность на уровне операций: до 10 RISC-подобных операций на одном ADSP за один цикл; масштабируемость - от одной 32-разрядной до четырех 8-разрядных операций в АЛУ; одно 16-разрядное или два 8-разрядных умножения в MAC)	TMS320C8x (TI)
8 (1996)	Множество функциональных устройств. ШП, 2 ШД, 3 АШ	Lucent 16xx, Atmel Lode
9 (1997)	Суперскалярная VLIV-архитектура (Veloci), суперскалярность на уровне операций - 8 за один цикл (6 АЛУ, 2 MAC, кэш-память программ, 4-канальный ПДП)	TMS320C6xx (TI)
10(2005)	Комбинированная двухуровневая фон-неймановская/реккуррентно-потоковая однокристальная многосекционная суперскалярная архитектура; четырехступенчатый конвейер команд; параллельность на уровне задач в РОУ- от 4 до 16 Секционных Вычислителей (СВ) на кристалле; параллельность на уровне операций: до 10 RISC-подобных операций на одном СВ за один цикл; каждый СВ содержит по одному: 16-разрядному АЛУ, 16-разрядному MAC-устройству, секционному 16-разрядному Аккумулятору, 40-разрядному сдвигателю, устройству округления и Преобразователю тегов) и два 40-разрядных MAC-аккумулятора	РОС на базе ПЛИС семейства Excalibur, Altera (ИПИ РАН)

Здесь:
ШП - шина программ (памяти программ); ШД - шина данных (памяти данных);
ШДЧ (З) - ШД для чтения (записи); ШДП - периферийная шина данных;
ШД_ПДП - шина данных для прямого доступа к памяти (ПДП) и к периферийным устройствам; АШ - адресная шина (для ШП/ШД/ШДЗ/ШДЧ/ШДП/ШД_ПДП);
ПЭ - процессорный элемент; MAC - multiplier with accumulation.

Алгоритмы обработки цифровых сигналов во многом сводятся к умножению с накоплением. Не случайно, что первый ЦСП (DSP-1, AT&T) так называемого нулевого поколения (см. строку 1 табл. 1) базировался именно на фон-неймановской архитектуре. Однако ЦСП относятся к классу вычислительных средств реального времени, которые должны обрабатывать непрерывный поток входных сигналов в темпе их поступления. Введением только аппаратных умножителей (MAC-устройств) в состав ЦСП эту задачу не решить.

На одноцикловую операцию собственно умножения приходится от трех до пяти подготовительных действий: обработка команды (минимум один цикл), чтение первого операнда, чтение второго операнда и (достаточно часто) запись результата умножения в память. Гарвардская архитектура, предложенная в рамках проекта ENIAC и нашедшая применение в ЦСП, была расширена по двум направлениям:

- общее пространство памяти данных физически реализовывалось в отдельных модулях памяти (как минимум - двух) с отдельными шинами данных и адреса (в табл. 1 приведено большое разнообразие числа и видов шин данных в поколениях ЦСП; главный отличительный признак ЦСП-архитектур - многошинность);
- стал возможен обмен между внутренними памятями программ и данных.

2) Простота и эффективность RISC-архитектур и, в первую очередь, исполнение всех команд за один такт (цикл) во многом соответствовали требованиям цифровой обработки сигналов (ЦОС) и были востребованы в ЦСП. Тем не менее, существенное различие RISC- и ЦСП-архитектур заключается в специфике их механизмов управления. Основа механизма управления быстродействующих RISC-архитектур - система прерываний, в том числе вложенных. В них на собственно цифровую обработку остается не так много вычислительных ресурсов. Основное отличие ЦСП определяется необходимостью соответствовать требованиям реального времени: использование механизма прерываний носит ограниченный характер, и разрешена только обработка событий, четко определенная по времени. Однако по мере повышения производительности ЦСП появляется понятие резервной (неиспользуемой) производительности, которая может быть использована для обработки прерываний и других задач, идущих не в реальном масштабе времени [3].

3) Резервная производительность используется для интеграции функциональных возможностей управления и обработки сигналов в одном ЦСП-устройстве. Функции, традиционно являющиеся атрибутами микроконтроллеров и RISC-микропроцессоров,

становятся регулярными для многих ЦСП. Введение в ЦСП средств использования различных периферийных интерфейсов обеспечивает поддержку системного уровня.

Заимствование решений идет в обе стороны. Фирма TI - лидер в области ЦСП - объявила о создании ЦСП-микроконтроллера TMS320C240x [4], интегрирующего достоинства двух этих функциональных классов. Фирма Microchip (США), традиционно специализирующаяся на производстве 8- и 16-разрядных микроконтроллеров, объявила о запуске в производство в 2002 году сразу трех семейств ЦСП-контроллеров dsPIC30F для управления двигателями - для эффективной обработки сигналов от датчиков и цифровых микроконтроллеров общего назначения. Фирмы Intel и ADI объявили о начале разработки совместного ЦСП под кодовым названием Frio, который, работая в связке с StrongARM-2 - XScale (также объявленном Intel), должен составить конкуренцию аналогичной паре из DSP TI TMS320C55x и процессора ARM9.

4) Отличительным признаком ЦСП является широкая аппаратная поддержка различных функций в виде отдельных функциональных узлов: АЛУ, умножителей, сдвигателей, декодеров, устройств сравнения и т.д. Эффективное использование этих аппаратных ресурсов возможно за счет организации их параллельной работы. Традиционно в ЦСП для этих целей широко использовались многофункциональные команды. В микропроцессорах одним из средств организации параллельной работы вычислительных ресурсов является VLIW-параллелизм (параллелизм на уровне команд с архитектурой VLIW). По существу, VLIW-параллелизм базируется на двух структурах - умножении с накоплением и многократных шинах, которые составляют основу ЦСП. Как только прогресс в технологии обеспечил экономическую целесообразность использования в ЦСП команд большой разрядности, VLIW-параллелизм был востребован в семействе TMS320C6xx, что обеспечило поднятие производительности ЦСП до 9000 MIPS.

5) Имеет место взаимопроникновение не только аппаратных, но и программных компонентов, что важно для понимания будущих тенденций развития ЦСП.

На протяжении полутора десятилетий доминирующим критерием эффективности ЦСП являлось соотношение "*стоимость/производительность*" ($/MIPS). Постепенно стал использоваться критерий "*стоимость/производительность/мощность потребления*" ($/MIPS/mW). Трудоемкость программирования, которая, в основном, ложилась на плечи потребителей, этим новым критерием также не оценивалась.

По мере усложнения архитектур ЦСП, повышения их мощности и расширения областей применения трудоемкость ассемблерного программирования, остававшегося до последнего времени доминирующим стилем программирования, становится основным препятствием быстрого использования функциональных возможностей предлагаемых аппаратных средств. В связи с этим наблюдается переход к критерию "*стоимость аппаратуры* (собственно ЦСП-кристалла)*/производительность/мощность потребления/стоимость программных средств*". По мере совершенствования архитектур ЦСП и технологии изготовления снижение стоимости разработки аппаратуры намного опередило снижение стоимости разработки программных средств. Поэтому в области ЦСП наблюдается перенесение акцента от аппаратуры к программному обеспечению: по экономическим соображениям уменьшение времени разработки ЦСП-систем для конкретной предметной области более оправдано, чем возможное недоиспользование потенциальных MIPS. Смене этого акцента способствуют следующие обстоятельства:

- резервная производительность во многих прикладных областях может быть использована для перехода на другой язык программирования - язык высокого уровня;
- усложнение архитектур ЦСП (до суперскалярных многопроцессорных систем на кристалле) делает невозможным использование сложившегося стиля программирования ЦСП - формирования программы на базе "задача за задачей" с детализированным анализом кода для каждого алгоритма в работе системы;
- переход на стандартные языки высокого уровня дает возможность использовать богатый задел, накопленный в области микропроцессоров общего назначения, и привлечь к разработке программного обеспечения для ЦСП-систем широкий круг программистов.

Приоритетным направлением разработки ЦСП-систем становится создание интегрированных сред (как программных, так и аппаратных) разработки и отладки программ; инструментальных средств поддержки языков высокого уровня; эффективных трансляторов с языков высокого уровня (в первую очередь, С-трансляторов); модульного программного обеспечения и программного интерфейса, гарантирующего 100-процентную переносимость программных модулей от независимых поставщиков.

Учитывая тенденции развития ЦСП, хорошую сочетаемость принципов ПА и требований со стороны ЦОС-применений (см. раздел 2.3), а текже потенциально высокие характеристики РОС, можно выдвинуть предположение, что комбинированный двухуровневый вариант реализации РОС (см. раздел 4.2) может претендовать на представительство следующего (десятого) поколения ЦСП.

2. Классификация ЦСП

К настоящему времени спроектированы и опробованы сотни различных ЦСП, использующих в своей архитектуре тот или иной вид параллельной обработки данных. Для того чтобы разобраться в обилии ЦСП, представленных на рынке, в особенностях их архитектурных решений и степени соответствия этих решений требованиям отдельных применений, необходимо систематизировать их определенным образом. Такое средство систематизации - классификация, в данном случае - разделение всего многообразия ЦСП на отдельные классы. Процесс классификации сугубо субъективный и определяется задачами, которые ставит перед собой его создатель.

2.1. Классификация ЦСП по диапазону обрабатываемых чисел

Динамический диапазон обрабатываемых данных в ЦСП зависит от двух особенностей: типа используемой арифметики и разрядности обрабатываемых операндов. Большую гибкость и точность результатов вычислений обеспечивает арифметика с плавающей запятой. Однако аппаратные затраты такой арифметики существенно повышают стоимость кристалла, что делает целесообразным ее использование только в специальных применениях. Именно поэтому 95 % используемых (покупаемых) моделей ЦСП представляют собой процессоры с фиксированной арифметикой.

На рис. 1 представлена классификация ЦСП по диапазону обрабатываемых чисел. Для сравнения там же представлены (менее подробно) и процессоры общего назначения.

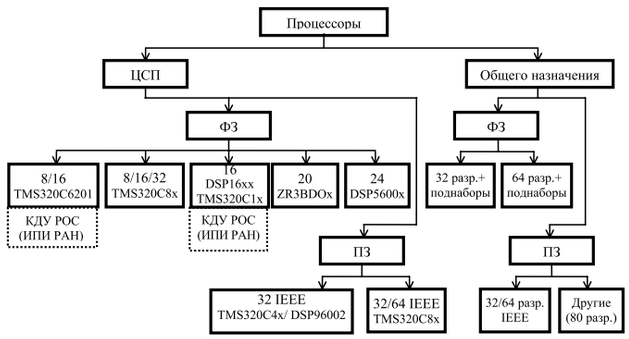

Рис. 1. Классификация средств ЦОС по диапазону обрабатываемых чисел (расширенный вариант классификации из [14])

Первое поколение ЦСП было представлено 16-разрядными ЦСП с фиксированной арифметикой. Однако на рис. 1 в крайней левой позиции фигурирует представитель семейства TMS320C6x - TMS320C6201, процессор с фиксированной арифметикой (на рынке с 1997 года), который представлен как *8/16-разрядный* процессор с фиксированной арифметикой. Это требует пояснения.

Любой 16-разрядный ЦСП обеспечивает обработку и 8-разрядных операндов. Однако при этом старшая половина процессорного блока ЦСП не используется. Поэтому пользователь, выбирая 16-разрядный ЦСП с позиций требуемого динамического диапазона для конкретного применения, при обработке 8-разрядных чисел использует его неэффективно, а доля 8-разрядных операций может составлять при этом 10 - 40 %. ЦСП TMS320C6201 обеспечивает эффективную обработку 8-разрядных данных - выполнение двух любых 8-разрядных операций в одном цикле. Именно это дает основание определять его как 8/16-разрядный ЦСП.

Семейство TMS320C8x обеспечивает эффективные вычисления с 8/16/32-разрядными данными - выполнение четырех 8-разрядных, двух 16-разрядных или одной 32-разрядной операции в одном цикле (метод распараллеливания - блочное расщепление).

На рис. 1 представлены также ЦСП с 16-, 20- и 24-разрядной фиксированной арифметикой. С увеличением разрядности увеличивается и диапазон возможных чисел.

Рассмотрим в качестве примера ЦСП фирмы Motorola - семейство DSP5600. Слово используется для представления 24-разрядных дробных чисел; старший 23-й разряд - знаковый [5]. Диапазон возможных чисел С: $-1 \le C_{(10)} \le 0{,}9999998 = 1 - 2^{-23}$, что обеспечивает динамический диапазон чисел при арифметических и логических операциях (исключая операцию умножения): $D = -20\lg 2^{-24} = 144$ [дБ]. Для сравнения, диапазон D при арифметических операциях TMS320C8x составляет 193 [дБ].

Статистический анализ С-моделей представительных задач, подлежащих решению на РОС [6]-[8], позволяет сделать вывод о целесообразности разработки двух классов РОС - 8/16- и 16-разрядный РОС с фиксированной арифметикой.

2.2. Классификация ЦСП по сферам применения

После того как пользователь выбрал определенный класс ЦСП, приемлемый с точки зрения диапазона чисел, дальнейшая конкретизация выбора требует использования других параметров. Из многих соображений, определяющих оценку уровня ЦСП, наиболее важными являются стоимость, производительность и потребляемая мощность (каждый из этих параметров, в свою очередь, комплексный, зависящий от ряда факторов). В соответствии с этими параметрами все ЦСП можно разбить на три класса: ЦСП общего назначения (со средневзвешенными значениями этих параметров); ЦСП высокой производительности и недорогие ЦСП малого потребления. В качестве примера на рис. 2 представлен расширенный вариант классификация ЦОС для беспроводных коммуникаций следующего поколения [9]. (С учетом заимствования и для однообразия будем использовать англоязычные единицы измерения.) В трех заимствованных классах этой классификации в качестве оценки производительности ЦСП используется характеристика MOPS (Million Operations Per Second).

Эта классификация полезна не только для пользователей; ее должны серьезно учитывать и разработчики ЦСП. Разрабатываемый ЦСП заранее ориентируется на определенный класс применений, и поэтому архитектурные решения должны сопоставляться с этими параметрами: цена аппаратуры должна сопоставляться с ее MIPS-эффектом и возможными потерями по потреблению питания.

Обычно принятие одного из вышеуказанных параметров в качестве экстремального означает некоторую степень потери в каждом из двух других. Однако использование ЦСП для быстродействующих систем типа многоканальных базовых станций одновременно требует разумного потребления мощности, чтобы иметь возможность упаковать множество каналов в одном ЦСП. И наоборот, маломощные применения типа беспроводных телефонов

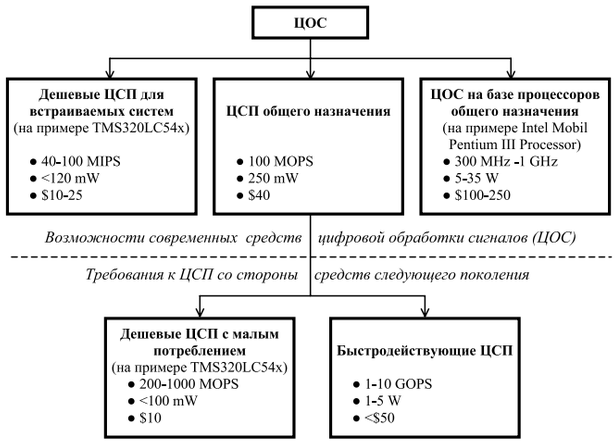

Рис. 2. Классификация средств ЦОС по сферам применения

требуют высокой производительности. Таким образом, для этих классов применений характерна не максимизация производительности или минимизация потребляемой мощности, а нахождение лучшего компромисса между этими параметрами для конкретного применения. Архитектуры TMS320C55x и TMS320C64x служат хорошей иллюстрацией такого компромисса [10].

Для оценки производительности ЦСП или, в общем случае, вычислительных средств используются разнообразные метрики: рабочая частота Hz, MIPS, MOPS (MPLOPS), различного рода эталонные тесты (benchmarks) и т.д. Каждая из используемых на практике метрик имеет свои особенности и свою сферу применения.

Для пользователя наиболее объективной оценкой производительности является время исполнения его программы - от ее инициализации до получения конечных результатов. Однако такая оценка возможна только на конечной стадии - реальном прогоне программы пользователя на реальной аппаратуре. На стадии проектирования необходимы предварительные оценки производительности вычислительных средств.

В области ЦСП общепринятой является оценка производительности в MIPS- (для фиксированной арифметики) или MPLOPS-параметрах (для арифметики с плавающей запятой). Большинство команд в ЦСП выполняется за один цикл; зная его длительность, можно перейти к MIPS-оценке, учитывающей структуру цикла, частотный вариант исполнения ЦСП и предполагающей исполнение команд в конвейере. Использование этой метрики носит ограниченный (конкретной сферой применения) и, в определенной степени, рекламный (максимально возможный) характер и имеет следующие особенности:

- такая оценка не может использоваться для сравнения ЦСП с разными наборам команд; для выполнения одной и той же программы на двух ЦСП с одинаковыми структурой цикла, длиной конвейера и рабочими частотами, но с разными наборами команд может потребоваться существенно разное число MIPS;
- выполнение двух различных программ с одним и тем числом команд на конкретном ЦСП также может показать разное число MIPS;
- MIPS-оценка может находиться в обратно пропорциональной зависимости с реальной производительностью ЦСП.

При соблюдении определенных условий в качестве более объективного параметра может выступать MOPS-оценка. Под операцией следует понимать при этом не компоненты команды (выборку команды, генерацию адресов операндов, выборку операндов, выполнение операции, округление результата и т.д.), как это принято

в среде разработчиков ЦСП, а обычные операции сложения, вычитания, умножения и деления. В рамках любого приложения могут быть представлены операции, требующие разной разрядности - 8, 16, 20, 24, 32, достаточно редко - 64, поэтому алгоритмическая сложность прикладных задач может быть оценена следующим образом:

$$N = a8 + a16 + a20 + a24 + a32 + a64 + ... + \\ + s8 + s16 + s20 + s24 + s32 + s64 + ... + \\ + m8 + m16 + m20 + m24 + m32 + m64 + ... + \\ + d8 + d16 + d20 + d24 + d32 + d64,$$

где N - число операций, a идентифицирует операцию "add", s - "subtraction", m - "multiplication", d - "division".

Если наряду с операцией $a16$ (доминирующей в рамках операций сложения в данной задаче) указывается операция a8, это означает, что имеется определенный *процент регулярно повторяющихся 8-разрядных сложений над определенным образом структурированными данными*, например, два 8-разрядных операнда в одном 16-разрядном слове. В тех ЦСП, которые поддерживают метод блочного расщепления для этого типа операций, эти две операции могут быть выполнены в рамках одной команды. Если же 8-разрядные операции выполняются над случайно расположенными данными, то их следует указывать как 16-разрядные операции.

От числа операций в программе можно перейти к MOPS-оценке:

$$MOPS = \frac{N}{t_{\max}} \times 10^6,$$

где N - число операций в программе, t_{max} - максимально возможное время выполнения программы (в секундах), которое определяется критическим циклом алгоритма.

Имея MOPS-оценку алгоритмической сложности конкретного алгоритма и зная архитектурные особенности конкретного ЦСП, пользователь может получить достаточно точную MIPS-оценку его реализации. И наоборот, зная MOPS-оценки набора алгоритмов некоторой прикладной области, разработчики ЦСП могут реализовать необходимые архитектурные особенности, обеспечивающие эффективную реализацию этих алгоритмов.

В процессорах общего назначения, где система команд существенно шире (отвечает требованиям общего назначения), метрика в виде рабочей частоты процессора является более объективным параметром, чем MIPS для ЦСП. Именно этим объясняется использование в рамках одной классификации (рис. 2) трех метрик: MIPS, MOPS и Hz.

На практике для ЦОС-обработки в мультимедийных применениях иногда используются процессоры общего назначения. В качестве примера на рис. 2 приведены параметры Intel Mobile Pentium Processor III, ориентированного на мобильные применения, в том числе на использование в сотовых и беспроводных телефонах. Разброс характеристик этого микропроцессора связан с выбранным частотным исполнением. В классе ЦСП, ориентированных на эти же применения, приведены параметры семейства TMS320LC54x. Если сопоставление данных по быстродействию средств ЦОС-обработки для разных классов применений требует дополнительного анализа, то сравнение по стоимости и потребляемой мощности достаточно очевидно.

В рамках рассмотренной классификации разрабатываемый РОС может быть предварительно отнесен к классу быстродействующих ЦСП, и его стоимостные показатели которого могут быть определены только при массовом производстве.

E.V.Krishnamurthy [11] предлагает использовать для классификации параллельных ВС четыре характеристики: степень гранулярности, способ реализации параллелизма, топологию связи процессоров и способ управления процессорами. Для каждой степени гранулярности рассматриваются все возможные способы реализации параллелизма; далее для каждого полученного таким образом варианта рассматриваются все комбинации топологии связи и способов управления процессорами. В результате получается дерево, в котором каждый ярус соответствует своей характеристике, каждый лист представляет отдельную группу компьютеров в данной классификации, а путь от вершины дерева однозначно определяет значения указанных выше характеристик (см. рис. 3).

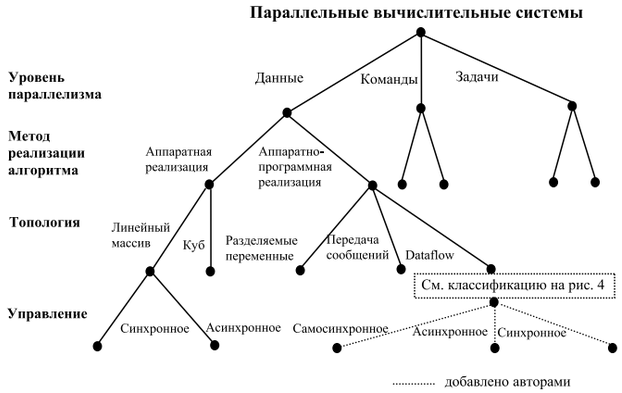

Рис. 3. Классификация Krishnamurthy

Первые два уровня практически точно повторяют классификацию А. Базу. Третий уровень классификации - топология связи процессоров - тесно связан со вторым. Если был выбран аппаратный способ реализации параллелизма, то надо рассмотреть топологию связи процессоров (матрица, линейный массив, дерево, и т.п.) и степень связности процессоров между собой (сильная, слабая или средняя), которая определяет долю накладных расходов при организации взаимодействия процессоров. При комбинированной реализации параллелизма, помимо степени связности, надо дополнительно учесть механизм взаимодействия процессоров: передача сообщений, разделяемые переменные или принцип *потока данных* (по готовности операндов).

Наконец, последний, четвертый уровень - способ управления процессорами - определяет общий принцип функционирования всей совокупности процессоров вычислительной системы: синхронный, потоковый или асинхронный.

На основе выделенных характеристик нетрудно определить место, например, потоковой архитектуры: гранулярность - на уровне команд; реализация параллелизма - комбинированная; связь процессоров - простая топология с сильной либо средней связностью и использованием потокового принципа; способ управления - потоковый.

Несмотря на то, что классификация E.V. Krishnamurthy построена лишь на четырех признаках, она позволяет выделить и описать такие "нетрадиционные" параллельные системы, как систолические массивы и потоковые машины.

С точки зрения способов управления процессоров в РОС, в частности, в двухуровневом варианте его реализации, предполагается использование синхронного способа на управляющем уровне РОС, самосинхронного - на уровне РОУ и асинхронного - на межуровневом взаимодействии.

2.3. Классификация потоковых архитектур по эффективности эксплуатации исходного алгоритмического параллелизма

Прежде чем классифицировать ПА, целесообразно ответить на два вопроса:

- в какой степени они соответствуют особенностям ЦОС-проблематики;
- каково состояние и перспективы развития этого направления.

С начала 80-тых годов, когда J.B.Dennis сформулировал принципы ПА [12], на рынке не появилось коммерчески выпускаемых изделий, целиком базирующихся на этих принципах. Тем не

менее, именно это направление в области разработки ВС нового поколения относится к наиболее перспективным. Многие исследовательские зарубежные центры (Японии, США, Франции и др.) достаточно интенсивно (при финансовой поддержке промышленных компаний) ведут исследования, разработки и испытания опытных образцов ВС на базе потоковой парадигмы. В течение ряда лет работы по созданию потоковой вычислительной машины с использованием оптической элементной базы (проект ОСВМ) велись в Вычислительном центре коллективного пользования Российской Академии наук [13].

Отдельные архитектурные аспекты потоковых машин, эффективность которых подтверждена результатами модельных и натурных испытаний, уже находят применение в современных средствах ВТ. В особенности это касается различных языковых средств программирования, которые первоначально создавались специально в рамках потоковой парадигмы и достаточно эффективно и широко используются в настоящее время. Программные потоковые спецификации широко применяются в области ЦОС [14]-[18].

В [19] приводятся данные о нескольких примерах разработки многопроцессорных систем, ориентированных на обработку сигналов и базирующихся на принципах ПА. Наиболее известные из них - Hughes Data Flow Multiprocessor, TI Data Flow Signal Processor и AT&T Enhanced Modular Signal Processor. В первых двух проектах задачи по отдельным процессорам распределяются во время компиляции, а в проекте AT&T EMSP - во время исполнения, т.е. динамически.

В каждой из этих машин используется достаточно сложная аппаратура для динамического распределения задач по процессорам и коммуникационная сеть для направления промежуточных результатов, выработанных одним ЦСП (источником), для их последующей обработки в других ЦСП (приемниках). Авторы отмечают, что для большинства приложений ЦСП динамическое распределение излишне: предсказуемость времени исполнения программы делает статические методы распределения жизнеспособными.

Другой пример использования принципов ПА в области обработки изображений - кристалл μPD7281 фирмы NEC [20]. Один кристалл содержит один функциональный блок, который может быть соединен с другими идентичными блоками для конвейерной обработки. Операторы на каждый отдельный процессор назначаются статически, а внутри процессора - динамически. Введены специальные команды для обработки изображений.

В [21]-[24] приводятся результаты разработки имитационной модели цифрового сигнального процессора обработки данных

(DFSP - Data Flow Signal Processor), результаты применения двух приложений и результаты аппаратного исполнения.

Модульная DPSP-архитектура обеспечивает на аппаратном уровне гибкость настройки на требования различных применений. Архитектура DPSP ориентирована на шину и применима в широкой области ЦОС, отвечающей определенным требованиям. Исключение составляют рекурсивные алгоритмы, реализация которых на DFSP неэффективна - их потоковые графы не обеспечивают достаточного уровня параллелизма.

Приведенные примеры дают достаточно оснований для утвердительного ответа на первый вопрос: принципы ПА и требования со стороны ЦОС хорошо сочетаются друг с другом в приложениях, характеризующихся большой степенью параллелизма. Основным сдерживающим фактором широкого практического использования такой интеграции является высокая цена многопроцессорных реализаций и потери производительности при внекристальной реализации коммуникационной межпроцессорной сети. Последние архитектурные и технологические достижения в области ЦСП дают основание предполагать, что парадигма потоковых вычислений, широко используемая в ЦСП на программном уровне, найдет адекватную и эффективную поддержку и на аппаратном уровне.

Внедрение потоковой парадигмы в область ЦСП будет происходить с учетом их особенностей: необходимости обработки большого массива входных данных в реальном масштабе времени; относительно низкой цены, малого энергопотребления и высоких надежностных показателей для удовлетворения требованиям встраиваемых систем. Поэтому наработки и архитектурные решения, накопленные за 20-летнюю историю развития потоковых архитектур, постепенно могут быть востребованы в области ЦСП. В связи с этим целесообразно перейти к классификационному анализу различных вариантов реализации вычислительных архитектур, базирующихся на принципе потока данных.

Существует большое разнообразие архитектурных решений и реализаций в области высокопроизводительных вычислительных средств - от однопроцессорных (со многими специализированными функциональными блоками) до многопроцессорных, многомашинных и сетевых конфигураций. Все они базируются на фон-неймановском типе процессора: используют программный счетчик как указатель последовательности исполнения команд. В связи с этим в [25] в отношении многопроцессорных конфигураций отмечаются две фундаментальные проблемы:

1) *Задержка при доступе к памяти* - время от момента

запроса на получение данных из памяти (нелокального модуля памяти, общего сетевого ресурса) до момента получения необходимых данных. Эта задержка может быть достаточно большой из-за неготовности данных (они еще не обработаны другими процессорными элементами). Переключение контекста на выполнение другого процесса, данные для которого имеются в локальной памяти, также связано с определенными временными потерями.

2) *Синхронизация процессов*, гарантирующая правильную очередность исполнения команд с учетом зависимостей по данным. Например, один процессор не должен выполнять чтение ячейки памяти, пока другой не произведет в нее запись.

В однопроцессорных конфигурациях, обрабатывающих входной массив, возникает необходимость убедиться в своевременном поступлении и обработке промежуточных результатов другими функциональными блоками, что связано с временными потерями.

Традиционные MIMD-процессоры для синхронизации работы в рамках решения отдельной задачи требуют явного описания фрагментов задачи, выполняемых на отдельном процессоре, и описания взаимодействия отдельных фрагментов [13], т.е. требуют от программиста извлечения параллелизма на низком уровне.

Потоковая парадигма, являющаяся одним из средств автоматического распределения вычислений между процессорами, не использует (по крайней мере, теоретически) явных средств синхронизации. Очередность выполнения команд в ней определяется не счетчиком команд, а готовностью операндов, необходимых для ее исполнения. Потоковые архитектуры, рассмотренные в предыдущем разделе как нечто общее, как и традиционные фон-неймановские архитектуры, предполагают различные формы ее реализации. На рис. 4 представлена классификация ПА по критерию степени реализации возможного алгоритмического параллелизма.

2.3.1. Статические потоковые архитектуры

Первая статическая потоковая архитектура (СПА) - MIT Static Dataflow Architecture - была предложена Jack Dennis [25]. Потоковая программа в СПА представляет собой полный набор так называемых шаблонов активности, каждый из которых однозначно соответствует вершине в потоковом графе и состоит из трех компонентов:

- *кода команды*, которая должна быть выполнена над операндами при их готовности (записи в поля операндов);
- *полей (слотов) операндов*, т.е. зарезервированных мест в памяти, куда должны быть помещены значения операндов;

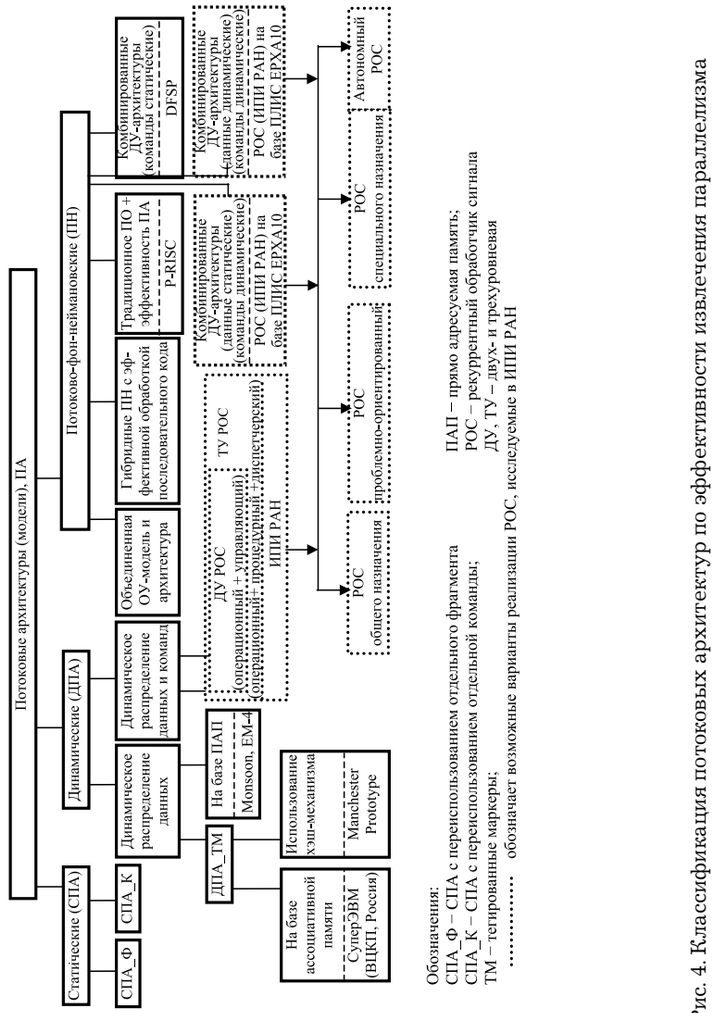

Рис. 4. Классификация потоковых архитектур по эффективности извлечения параллелизма

- *полей приемников* результатов выполнения команды (адресатов), которые должны их получить для дальнейшей обработки;
- *полей приемников* подтверждения выполнения команды и факта использования исходных операндов.

Пример структуры шаблонов активности и потоковой программы для вычисления значения $z = (x * y)^2$ приведен на рис. 5 (при изложении статической потоковой модели СПМ ряд положе-ний и

рисунок взят из [26]). Поле операнда 2 используется только при двухоперандных командах, например, $a + b$. В качестве операнда 2 может фигурировать константа. Команды (вершины в потоковом графе) инициализируются к исполнению тогда и только тогда, когда на всех ее входных дугах имеются маркеры, индицирующие готовность (приход) необходимых операндов. После исполнения команды маркеры из входных дуг изымаются и помещаются на выходную дугу. Для обнаружения факта прихода операндов в его поле вводится дополнительный разряд наличия операнда.

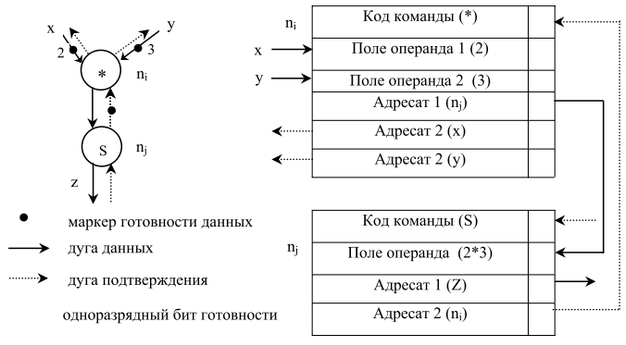

Рис. 5. Структура шаблона активности в СПМ

Статическими подобные архитектуры называют потому, что имеется *статическое* ограничение на дуги: на одной и той же дуге в одно и тоже время может быть только один маркер, т.е. имеет место *статическое распределение памяти*. Это означает, что пока инициированная команда не будет выполнена (пока на ее выходной дуге не появится маркер), новые значения операндов из того же шаблона активности не могут создать условия для повторного запуска той же (еще не законченной) операции. Для реализации этого правила в аппаратуре необходимо введение специальных сигналов *подтверждения* (дуг подтверждения), которые возвращаются в шаблоны активности - источники операндов, инициировавших выполнение команды. На практике в шаблоны активности возвращаются маркеры подтверждения. Из рис. 5 видно, что поток маркеров подтверждения в СПА создает серьезную нагрузку на коммуникационные пути.

Достоинства статической модели:

- простота реализации потокового графа непосредственно на аппаратуре;

- отсутствие динамического распределения памяти;
- отсутствие многословных буферов между операторами (буфер одновременно может хранить не более одного маркера).

Основной недостаток - ограниченный параллелизм по данным.

Ограничение возможности использования параллелизма по данным в СПА объясняется отсутствием в ней явной поддержки повторно используемых фрагментов программы. Например, оператор цикла указывает на многократное использование тела цикла, причем точное число использований заранее (статически) не всегда известно, а определяется в ходе выполнения программы (динамически) [13]. Обращение к процедуре (функции) - другой пример многократного ее исполнения с разными исходными данными.

Если на функциональном уровне различные итерации цикла не связаны зависимостями по данным, то они могли бы выполняться параллельно (с учетом числа имеющихся вычислительных ресурсов). Однако на аппаратном уровне в СПА нет средств, чтобы отличить данные одного поколения повторного использования фрагмента кода от данных других поколений (например, принадлежащих разным итерациям цикла).

Введение дополнительных (искусственных) зависимостей по данным разных поколений - один из путей решения этой проблемы. Например, пока не завершатся вычисления над данными первого поколения, не могут начаться вычисления над данными второго поколения; повторно используемые фрагменты программы должны выполняться строго последовательно. При этом принцип потоковых вычислений соблюдается только в рамках *отдельного фрагмента* (СПА-Ф): в любой момент повторно используемые участки программы могут использоваться только одним поколением данных [13].

Другой, более эффективный (с точки зрения использования параллелизма по данным) вариант реализации СПА предполагает расширение принципа потока данных до *повторно используемой команды* (СПА-К). При этом любой отдельный фрагмент кода может быть использован разными поколения данных при условии, что любая команда фрагмента не может быть инициирована более чем одним поколением операндов.

Рассмотрим более подробно особенности реализации СПА, обобщенная структура которой приведена рис. 6,а.

Наиболее существенным с позиций потоковой концепции является функциональный блок "Память-синхронизатор", который осуществляет прием новых маркеров, их сопоставление с содержимым шаблонов активности и, если все необходимые для выполне-

ния команды операнды присутствуют, ввод адреса соответствующего шаблона в блок "Очередь команд" (каждый шаблон имеет свой уникальный адрес). Очередь команд представляет собой буферное FIFO. Блок выборки считывает первый адрес шаблона из "головы" FIFO, а содержимое шаблона - из памяти шаблона, и направляет их в АЛУ для обработки. Пакеты результатов направляются в блок "Вывод", который (в зависимости от адресата) посылает их в коммуникационную сеть или возвращает в локальный блок "Ввод". Критический параметр такой организации СПА - пропускная способность коммуникационной сети.

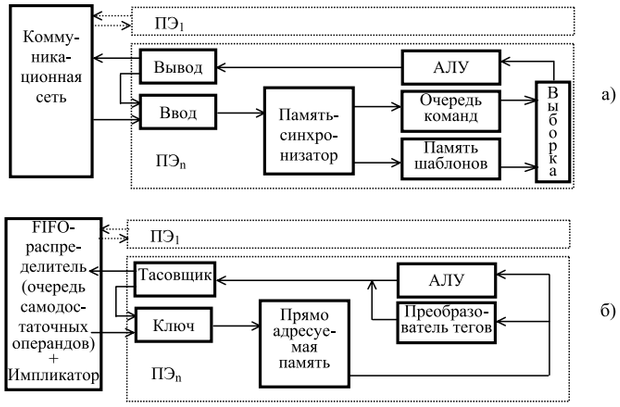

Рис. 6. Структура потоковых архитектур
а) стандартная статическая потоковая архитектура
б) потоковое рекуррентное операционное устройство

Указанные недостатки СПА могут быть несущественны для ЦСП в определенных областях применения. Достаточно часто поколения (порции) данных поступают в ЦСП на обработку последовательно. При этом на время обработки предыдущего поколения данных накладываются жесткие ограничения; по крайней мере, число поколений данных, которые с позиций реального времени могут обрабатываться параллельно, невелико. С учетом этого обстоятельства использование элементов СПА в ЦСП может быть продуктивно, тем более что аппаратные, а иногда и временные затраты на их реализацию существенно меньше, чем в динамических потоковых архитектурах (ДПА).

Недостатки, присущие СПА [27]:

- Ограниченные возможности эксплуатации параллелизма, в особенности параллелизма по данным. Реализации сигналов (маркеров) подтверждения обеспечивает только частичное совмещение итерационных циклов. Даже при отсутствии зависимостей по данным возможна реализация только конвейерного эффекта.
- Отсутствие средств явной поддержки процедур вызова и рекурсий (повторно используемого кода). Существует только одна копия набора шаблонов активности, и только один маркер может иметь место на одной дуге в каждый момент времени.
- Поток маркеров подтверждения увеличивает общий трафик маркеров, что может быть серьезной проблемой в мультипроцессорных системах.

Рекуррентно-динамическая парадигма (РДП) является основой построения спектра классов архитектур (см. рис. 4). Рекуррентная основа каждой из архитектур обеспечивает им определенные преимущества перед другими представителями данного класса.

Для сравнения на рис. 6,б приведена структура РОУ, которая может быть сопоставлена с СПА. На рис. 4 этот вид РОУ является элементом двухуровневой (ДУ) архитектуры в комбинированном (данные статические / команды динамические) или чисто рекуррентном варианте исполнения. Из рис. 6,б видно, что в структуре РОУ отсутствует Память шаблонов, поскольку вся необходимая информация по обработке операндов содержится в их функциональных полях. Наоборот, с функциональной точки зрения можно считать, что в РОУ имеет место Память шаблонов не на одну копию шаблонов активности (узкое место СПА), а на бесконечно много копий - по числу пар операндов, пришедших на обработку в АЛУ.

Прямо адресуемая память (ПАП) в РОУ в функциональном плане эквивалентна Памяти-синхронизатору: осуществляет прием новых маркеров, их сопоставление с содержимым ПАП и, если присутствуют все операнды, необходимые для выполнения команды, выдачу содержательной части операндов на обработку в АЛУ, а функциональной части - на обработку в Преобразователь тегов. Если пришедший операнд не находит свою пару в ПАП, то он временно сохраняется в ней. В области ЦСП применений ПАП, в основном, выполняет функцию синхронизации (управления). Результаты прогона на программной модели РОУ набора представительных алгоритмов в области речевой технологии показали, что требуемый объем ПАП не превышает 32 слов.

Распределитель (FIFO-буфер) в РОУ выполняет функции коммуникационной сети (распределяет операнды между процессорными элементами) и Очереди команд в СПА. С другой стороны

он может выполнять функцию многооперандного буфера в динамических ПА для реализации многих маркеров на одной дуге. Архитектура устройства РОС и его основной вычислительной части РОУ содержит аппаратно-программные средства для поддержки повторно используемых фрагментов программ при реализации капсульного стиля программирования. При этом отпадает необходимость формирования сигналов подтверждения.

Результат анализа показывает, что устройство РОУ по сравнению со стандартной СПА характеризуется существенно большей эффективностью извлечения параллелизма, отсутствием временных потерь на передачу сигналов подтверждения и меньшими аппаратными затратами. С учетом особенностей применения РОС вариант его реализации, показанный на рис. 6б, следует рассматривать как предпочтительный.

2.3.2. Динамические потоковые архитектуры

Их основное отличие от СПА состоит в обеспечении параллельного выполнения различных итераций цикла и вызовов процедур. Это достигается введением тегированных маркеров (tagged tokens), которые позволяют различать номер итерации и контекст процедуры. Таким образом, выполнение команды разрешается, как только на каждом из ее входов (на входах соответствующего оператора в потоковом графе) присутствуют маркеры с идентичными тегами. Это правило инициации команды в ДПА устраняет необходимость использования сигналов подтверждения, уменьшая, таким образом, трафик маркеров и повышая степень извлечения параллелизма.

Формат тега: $t = c \bullet s \bullet i$, где: c - контекст вызова процедуры или вызова цикла, s - идентификатор команды, i - номер итерации цикла.

Контекст - динамический идентификатор уникальных поколений данных, аналогичный совокупности значений регистров фоннеймановских машин, однозначно определяющих вычислительную обстановку. В [13] указывается, что поля контекста в рамках проекта ОСВМ используются как для реализации параллельного выполнения повторно используемых кодов программы, так и для обработки составных данных.

а) Динамические потоковые архитектуры с тегированными маркерами

Центральным устройством ДПА_ТМ является Память сравнения тегированных маркеров (waiting/matching unit). Она определяет основные достоинства и недостатки этого вида реализации

ДПА. Память сравнения - специализированный вид организации памяти, обеспечивающий сравнение тегированных маркеров, которые ожидают в ней своих партнеров, с маркерами, поступающими из Очереди маркеров (буфера FIFO). В большинстве экспериментальных ДПА_ТМ память сравнения функционирует следующим образом. На теги, поступающие в Память сравнения, вычисляется хэш-функция, и вычисленное значение используется как адрес для доступа в параллельную хэш-таблицу.

Возможна другая форма реализации Памяти сравнения - в виде ассоциативной памяти. В этом случае маркер, поступающий из Очереди маркеров, сравнивается не с одним маркером по хэш-адресу, а со всеми маркерами, хранящимися в Памяти сравнения.

Идея ДПА была апробирована в ряде проектов (от чисто модельных экспериментов до реализации экспериментальных образцов); см., например, работы [28]-[30].

Недостатки, присущие ДПА_ТМ:

- Аппаратная сложность и время сравнения тегированных маркеров в Памяти сравнения достаточно высоки. Максимально возможная производительность архитектуры достижима при реализации Памяти сравнения в виде ассоциативной памяти. Но большой объем памяти, необходимый для хранения маркеров, делает этот подход нерентабельным.
- Степень параллельности в выполняемых программах не контролируется. Ряд приложений может генерировать больше параллельных действий, чем позволяют аппаратные возможности. Результат - снижение реальной производительности.
- Последовательные участки кода исполняются неэффективно. Потери времени на этих участках из-за сравнения тегированных маркеров, их обмена, формирования и доступа к различным структурам ЗУ не всегда могут быть скомпенсированы одновременной обработкой потоков данных на параллельных участках кода.
- Отсутствие ограничения числа одновременно выполняемых итераций цикла и контекста процедуры ведет к увеличению размера тегов, потерь времени на их коммуникацию по сети и объема аппаратуры.
- Невозможно использовать регистровые структуры с такой же эффективностью, как в обычной многопроцессорной системе.

Для снятия части указанных проблем был предложен другой подкласс ДПА - так называемые архитектуры с прямо адресуемой памятью (ДПА_ПАП).

б) Динамические потоковые архитектуры с прямо адресуемой памятью

Основная идея ДПА_ПАП-архитектур состоит в использовании прямо адресуемой памяти в качестве Памяти сравнения тегированных маркеров. При этом для каждой активной итерации цикла и обращения к процедуре выделяется отдельная область памяти (фрейм). Каждый фрейм используется для поддержки операндов в рамках отдельной команды. К фреймам можно обращаться, используя контекст как адрес доступа к фрейму. Другой практический прием связан с введением k-ограниченных активных циклов: число одновременных активных итераций цикла управляется (ограничивается), чтобы число размещенных в ПАП фреймов не было слишком большим.

Основные недостатки ДПА_ПАП-архитектур [31].

1) Низкая производительность реализации последовательного кода: результат текущего шага должен быть обработан на следующем шаге вычислительного процесса (непосредственная зависимость по данным на каждом шаге последовательного кода). В рассмотренном выше восьмиступенчатом конвейере выполнение инструкций одного и того же потока потребует более восьми циклов. Коэффициент использования потокового процессора - одна восьмая от его максимальной производительности.
2) Потери, связанные со сравнением маркеров: для выполнения двухоперандной команды должны иметь место два маркера; первый маркер, запоминаемый в Памяти маркеров, вводит в конвейер так называемый "пузырь" - пустоту. Результаты прогонов программ на Monsoon показали, что процент "пустот" в конвейере достигает 28,75 %.
3) Не используются возможности регистров.

Ряду авторов возможное решение проблем видится в комбинации потоковых (dataflow) и фон-неймановских (control-flow) механизмов.

3. Комбинированные потоково-фон-неймановские архитектуры

Интерес к комбинированным (гибридным) потоково-фон-неймановским (ПН) архитектурам возник практически одновременно с появлением "чистой" потоковой парадигмы. Гибридные архитектуры разрабатывались в стремлении объединить положительные особенности фон-неймановских и потоковых архитектур. По мнению авторов [32], кроме общеизвестных недостатков фон-неймановской организации вычислений (доминирующей в проектах

общего назначения), "...одной из ее главных сил ... является универсальный характер ее программной организации. Она универсальна не только в смысле машины Тьюринга, но способна эффективно поддерживать спектр стилей программирования и эффективно моделировать многообразие структур алгоритмов".

3.1. Объединенная одноуровневая ПН-модель

В [32] рассматривались три главных класса-конкурента организации программ: поток данных, многопоточное управление на базе фон-неймановских принципов и различные типы редукций. Авторы утверждают, что каждая модель подходит для представления частного класса языка программирования из-за способа, которым он передает информацию данных и управления. Для получения верной организации программы общего назначения необходимо интегрировать концепции, лежащие в основе этих трех моделей. В настоящей работе описываются результаты разработки модели организации программы для параллельной компьютерной архитектуры, управляемой данными, которая интегрирует концепции чистых вычислений потока данных с концепцией вычислений многопоточного управления.

Чтобы преодолеть отсутствие некоторых математических свойств в обычных языках программирования, опирающихся на фон-неймановскую организацию, были предложены новые, более простые формы программирования. На базе Объединенной модели реализована простая многопроцессорная система, поддерживающая модель [33].

Тем не менее, в рассмотренных работах не приводится каких-либо расчетных или экспериментальных данных, подтверждающих эффективность предложенной модели.

3.2. Гибридная ПН-модель с эффективной обработкой последовательного кода

Как уже отмечалось, одной из проблем ДПМ_ПАП является низкая производительность реализации последовательного кода. На коротких участках последовательного кода, которые имеют все необходимые данные в локальной высокоскоростной памяти (регистрах и кэш), любые накладные расходы на синхронизацию - расточительность. В связи с этим в большинстве гибридных моделей добиваются сокращения издержек, применяя некую форму кластеризации: некоторые последовательности узлов объединяются в потоки, которые выполняются последовательно и не несут расходов, связанных со сравнением маркеров.

Какие-то из этих гибридных технологий [34] сохраняют поня-