УДК 004.272.44

Рекуррентная потоковая архитектура: особенности и проблемы реализации

Ю.А. Степченков, Ю.Г. Дьяченко, Д.В. Хилько, В.С. Петрухин

Институт проблем информатики Федерального исследовательского центра "Информатика и управление" Российской академии наук

ystepchenkov@ipiran.ru, diaura@mail.ru, dhilko@yadenx.ru, cokrat2@rambler.ru

***Аннотация* — Представлены результаты разработки многопроцессорной рекуррентной потоковой архитектуры (МПРА), ориентированной для эффективного исполнения параллельных алгоритмов в области цифровой обработки сигналов (ЦОС). Показаны принципиальные отличия МПРА от существующих традиционных и нетрадиционных компьютерных архитектур, которые позволяют почти вдвое уменьшить время обработки инструкций за счет использования единого самодостаточного потока обрабатываемых данных и рекуррентного способа его представления. Приведен перечень дополнительных механизмов, которые позволили повысить производительность вычислений для целого ряда алгоритмов ЦОС. Некоторые из предложенных механизмов могут быть также использованы и в системах ЦОС традиционной архитектуры.**

***Ключевые слова* — потоковая архитектура, рекуррентность, цифровая обработка сигналов, память совпадений, суперскалярность.**

I. Введение

Актуальность разработки вычислительных архитектур на базе потоковой (data-flow) парадигмы не вызывает сомнений, поскольку потенциально они могут обеспечить существенно большую производительность по сравнению с традиционной фон-неймановской архитектурой. Тем не менее, эффективная реализация потоковых архитектур (ПА) наталкивается на целый ряд серьезных проблем, таких как: реализация рекурсии, циклов, итераций, работа с константами и др.

В ряде зарубежных экспериментальных проектов, проводившихся с начала 1980-х до середины 2000-х годов, в которых разрабатывались динамические потоковые (ДП) архитектуры, не нашлось эффективных путей решения перечисленных проблем [1-3]. Недостатки, присущие ДП-архитектурам:

- 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. Однако большой объем памяти, необходимый для хранения ожидающих маркеров, делает этот подход проблематичным.

- Степень параллельности в выполняемых программах не контролируется. Ряд приложений может генерировать больше параллельных действий, чем позволяют аппаратные возможности архитектуры. 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 запоминающего устройства (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.

- Большое разнообразие типов памяти – память совпадений, память переполнения (блок отложенных маркеров), память программ и констант, память очереди тегированных маркеров и другие – усложняет управление памятью.

- Невозможно использовать регистровые структуры с такой же эффективностью, как в обычной многопроцессорной системе.

Эффективное решение этих и других не указанных проблем возможно только на базе быстрой и емкой ассоциативной памяти, которая при этом будет наиболее аппаратно- и энергозатратным ресурсом, что целесообразно только для систем массового параллелизма.

В России серьезные результаты в области разработки универсальных высокопроизводительных систем, использующих принцип управления потоками данных, были достигнуты коллективом под руководством академика Бурцева В.С. (см., например, [4]). После кончины академика Бурцева В.С. и перехода большей части его коллектива из ИПИ РАН в ИППМ РАН работы по совершенствованию потоковой архитектуры были продолжены уже там.

Попытки использования потоковой парадигмы в области цифровой обработки сигналов также имеют многолетнюю историю [5-8]. Авторы указанных работ отмечают, что принципы ПА и требования со стороны алгоритмов ЦОС хорошо сочетаются друг с другом в приложениях, для которых характерна высокая сте-

пень внутреннего параллелизма. Основными сдерживающими факторами широкого практического использования такой интеграции являются высокая цена многопроцессорных реализаций и потери производительности при внекристальной реализации коммуникационной межпроцессорной сети. Для большинства приложений ЦОС динамическое распределение данных излишне, т.к. предсказуемость времени исполнения программы обеспечивает жизнеспособность статических методов их распределения.

В Институте проблем информатики Федерального исследовательского центра «Информатика и управление» РАН была разработана концепция принципиально новой многоядерной потоковой рекуррентной архитектуры (МПРА). Архитектура 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, но построенной на другом принципе, а именно – на принципе самодостаточных, рекуррентно сжатых данных (РД).

II. Особенности организации вычислительного процесса в МПРА

Предлагаемая архитектура нетрадиционна и радикально отличается по основным моментам не только от классической архитектуры фон Неймана, но и от других нетрадиционных параллельных архитектур, в частности, от архитектур, работающих по принципу потока данных (data-flow).

Для существующих традиционных и нетрадиционных компьютерных архитектур характерно наличие двух потоков: потока инструкций и потока данных.

В МПРА оба потока сливаются в один общий поток самодостаточных данных, в котором, помимо собственно обрабатываемых данных, содержится и необходимая для их обработки управляющая и служебная информация. Наглядное представление о сравнительных качествах рассматриваемых архитектур дает их сопоставление на рис. 1 и 2 (приводятся для наглядности):

- по организации памяти и инициации вычислительного процесса (рис. 1);

- по количеству шагов, необходимых для выполнения инструкции (рис. 2).

Инициатором выполнения ВП в фон-неймановской архитектуре является поток инструкций, извлекаемый из памяти программ вычислительным устройством (ВУ), – первичный поток инструкций (цифра 1 на рис. 1а). Программа-инициатор процесса привлекает данные для обработки: вторичный поток (цифра 2 на рис. 1а). Программа хранится в памяти инструкций в полном объеме и в статическом виде (отсюда название CF/S – Control Flow/Static). При этом существует проблема определения момента готовности данных для их обработки.

Рис. 2. Выполнение инструкций в сравниваемых архитектурах а) – в) – идентичны значениям на рис. 1

Для потоковых архитектур (DF – Data-Flow) сохраняется разделение ресурсов памяти на две области. Однако статус памяти данных меняется – из пассивной (вторичной) она превращается в активную (первичную) – цифра 1 на рис. 1б – в ячейках которой хранятся данные (операнды) с дополнительными функциональными полями (тегами). Как правило, функциональные поля содержат в себе информацию об исполнительном адресе инструкции (микрокоманды), которая должна быть извлечена из памяти программ для выполнения требуемых действий над поступившими на обработку компонентами пары (цифра 2 на рис. 2б). Полный объем привлекаемых инструкций также хранится в статическом виде.

МПРА относится к классу потоковых архитектур, причем в отличие от них тегируемые данные являются самодостаточными, т.к. они содержат всю информацию по их обработке: информацию о компоненте пары по обработке; о команде, которая должна быть выполнена над парой, и о месте назначения результата опе-

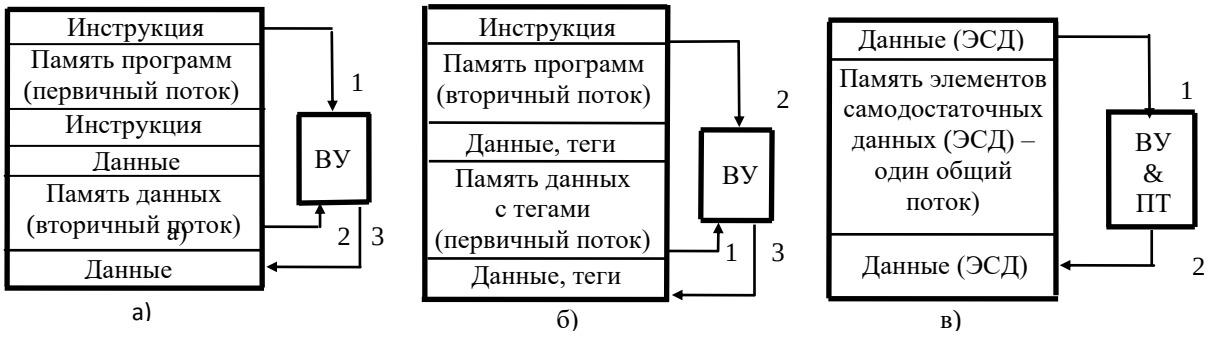

Рис. 1. Принципиальные отличия сравниваемых архитектур а) традиционная (CF/S), б) потока данных (DF/S), в) МПРА (DF/SD)

рации. Результат операции может быть: оставлен в конкретном ВУ для последующего использования, передан другим ВУ по коммуникационной сети, сохранен в памяти ЭСД в качестве промежуточного результата для повторного использования, передан во внешнюю среду в качестве окончательного результата.

Данные являются самодостаточными не только для текущего шага обработки. Они также содержат начальную сжатую информацию, которая определяет содержание тегов результата в процессе их рекуррентных преобразований, в том числе с учетом вариативности условий формирования этого результата.

В разрабатываемой архитектуре в состав ВУ включено автономное устройство преобразования тегов (ПТ), которое обладает возможностью саморазвертки рекуррентного ВП. Устройство ПТ инициируется операндами, пришедшими на обработку в ВУ, работает параллельно с ВУ и определяет действие на следующем шаге ВП (модифицирует теги). Устройство ПТ представляет собой относительно простую (по аппаратным затратам) комбинационную схему, содержащую средства настройки (в необходимых случаях) на предметную область.

В памяти МПРА нет исполняемой программы в традиционном смысле. Есть только начальные значения тегов операндов, которые динамически подвергаются рекуррентной саморазвертке устройствами ПТ (отсюда название архитектуры DF/SD – Data Flow/ Static Dynamic). Для выполнения алгоритма необходимо задать начальные значения функциональных тегов. Представленная подобным образом в МПРА программа была названа капсулой.

Для получения искомого результата в CF/S выполняется последовательность действий 1-5 (см. рис. 2). Число шагов в DF/S не меняется. В архитектуре DF/SD оно уменьшается до логически необходимого минимума – 3; уменьшается также суммарный объем требуемых ресурсов памяти и число пересылок между функциональными устройствами ЭВМ. Если уменьшение числа шагов в МПРА до 4 – очевидно: необходимость выборки инструкции отсутствует, то возможность совмещения в одном шаге сравнения тегов и дешифрации инструкции будет показана ниже при рассмотрении структуры рекуррентного обрабатывающего устройства (РОУ).

III. Комбинированнный двухуровневый вариант реализации МПРА

Исторический опыт разработки компьютерных архитектур свидетельствует [9], что новая архитектура может потерпеть неудачу по следующим причинам:

• если она базируется на чрезвычайно сложных аппаратных механизмах для поддержки предназначенной ей модели программирования;

• если она игнорирует совместимость с существующими вычислительными средами;

• если она игнорирует проблему программируемости.

Результаты работ по различным вариантам исполнения РОУ убедительно свидетельствуют, что она не требует сложных аппаратных механизмов для поддержки присущей ей капсульной модели программирования. Что касается второго положения, то характер работ в рамках РОУ и не предполагает достижения совместимости с существующими вычислительными средами. Наоборот, он предполагает разработку уникальной вычислительной среды, в максимальной степени учитывающей специфику МПРА.

Поиск компромиссного решения, включающего в себя совместимость с существующими вычислительными и аппаратными средами, возможен при удовлетворении определенных требований. В первую очередь это требование поддержки потокового характера вычислительного процесса, реализуемого в РОУ.

Реализация такого подхода возможна на базе гибридного варианта рекуррентного обработчика сигнала (ГАРОС): процессора с сокращенным набором команд NIOS2 и ПЛИС с организацией Programmable Logic Device (PLD) в рамках семейства микросхем фирмы Altera под условным названием Stratix IV [10] (см. рис. 3).

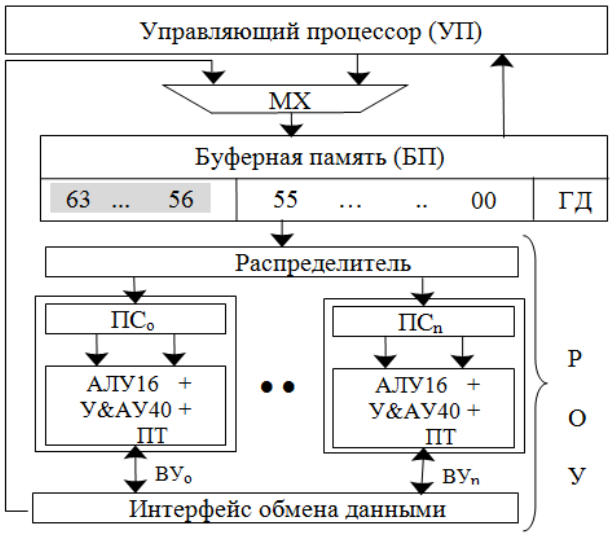

Рис. 3. Структурная схема ГАРОС ПС – память совпадения; ГД – готовность данных; РОУ – рекуррентное обрабатывающее устройство; ПТ – преобразователь тегов; У – умножитель; АУ – арифметическое устройство

В структуру ГАРОС входят: управляющий процессор (УП) на базе фон-неймановской архитектуры, распределитель, четыре однотипных рекуррентных вычислительных устройства (ВУ) и интерфейс межпроцессорного обмена – исполнительный уровень гибридного РОС. В свою очередь, каждое ВУ состоит из: памяти совпадения (ПС), вычислителя (на базе 16-разрядного АЛУ и умножителя с накоплением МАС) и преобразователя тегов (ПТ).

В качестве устройства сопряжения УП и РОУ (поток данных между ними носит интенсивный характер) предлагается использовать специальный тип двухпортовой буферной памяти (БП), реализованной в ПЛИС. Одновременное чтение капсулы из одного порта и за-

пись результатов операции в другой порт предположительно позволит сбалансировать поток данных между управляющим процессором и четырьмя операционными процессорными устройствами.

На управляющий уровень ГАРОС возлагается выполнение процедур предварительной подготовки капсул, реализация последовательных частей исполняемой программы и запись результатов в один порт БП. Операционный уровень РОС обеспечивает считывание капсул, готовых для исполнения; параллельные вычисления в ВУ и запись результатов вычислений в другой порт. Введение в состав РОС двухпортовой БП позволит также исключить потери времени на перемещение данных между двумя уровнями архитектуры ГАРОС.

Буферная память предназначена для хранения шаблонов капсул. В режиме обмена данными с РОУ происходит одновременное чтение и запись (со стороны РОУ) в БП благодаря двухпортовой реализации ЗУ.

Память признаков (готовность данных, ГД) контролирует наличие данных в шаблонах капсул. Если данные в БП готовы для считывания, то в бит ГД по данному адресу заносится '1', иначе – '0'. При обмене данными БП с РОУ возникает необходимость одновременного изменения признаков по двум адресам (обнуление для считанных данных и выставление '1' для принятых). Двухпортовая организация БП обеспечивает одновременную запись в порт 1 и чтение из порта 2 или одновременную запись в порт 1 и порт 2.

Максимально допустимая длина цикла работы БП – 4 обращения к памяти – время работы самой медленной ступени конвейера РОУ. Исходя из разрядности всего операнда (56 бит), количества операндов, которые необходимо считать в РОУ за один цикл (4), максимально допустимого количества обращений к памяти (4) и количества операндов, которые должны быть записаны в БП со стороны РОУ (4), БП разделена на восемь банков (Б0 – Б7).

Капсула может поставлять, а РОУ – воспринимать четыре типа операндов (табл. 1). Тип операнда указывается в обязательном для всех операндов поле [t].

Таблица 1

Типы операндов РОУ

t~поле		**Тип операнда**
Символ	**Код**	
E:	0	Пустой
A:	1	Вспомогательный
C:	2	Управляющий
D:	3	Содержательный

Только *содержательные* операнды содержат данные (отсюда ***D***-тип), подлежащие обработке в процессе вычислений. Этот тип операндов обрабатывается и воспринимается всеми модулями РОУ.

Пустой операнд (***E***-тип) не имеет структуры, не несет информации и не обрабатывается в РОУ.

Вспомогательные операнды (***A***-тип) содержат служебную информацию, не оказывающую влияния на организацию и ход ВП. Минимальный набор вспомогательных операндов: *терминатор капсулы*, *терминатор данных*, *глобальный конфигуратор* (конфигуратор РОУ в целом), *инициализатор* и *шаблон.* Каждый из них имеет свой специфический формат и предназначен для настройки одного из блоков РОУ.

Управляющие операнды (***C***-тип) управляют ходом ВП, координируют работу и настраивают некоторые модули структуры. Примеры операндов ***C***-типа: *конфигуратор отдельной секции* выполняет в необходимых случаях также функцию настройки преобразователей (ПТ); *тормоз* выполняет функцию программной остановки; *погонщик* реинициализирует ВП; есть еще операнды безусловного и условных переходов. В архитектуре ГАРОС реализован капсульный стиль программирования, где для каждого алгоритма разрабатывается шаблон капсулы, общий вид которой представлен в табл. 2.

Таблица 2

Структура капсулы

№	**Операнд**	**Структура**	**Содержание**
1	***Ani***	101%Ncpx@Ntvad	Идентификатор входной капсулы
2	***Adi***	*131%Nc1*	Идентификатор набора данных (НД0 и НД1)
3	***Acg***	*151%Cxpcres*	Глобальный конфигуратор
4	***Ai***	*160%I0nis%I1nis*	Инициализатор НД0 и НД1
5	***Aso***	*110%Stn[OcuSmDrse]*	Стартер выходных данных
6	***Atm (0-3)***	*161%Tcuhxmrse[OuShmD_]*	Шаблонный операнд секций 0-3
7	***Ccs***	*200@Cjlder[0cutShmD0001se]*	Конфигуратор секций 0-3
8	***Di***	*32@sV0_[OcutShmDrse]*	Содержательный операнд V0
9	***Di***	*32@sV1_[OcutShmDrse]*	Содержательный операнд V1
10	***Asi***	*111%St[OcutSmDrse]*	Стартер входных операндов
11	***Di2 + Di5***	*32V2_Sh_V3_Sh*	Содержательные операнды V2,V3, V4 и V5
12	***Di(n-1) + Di***	*32Vn-1_Vn-2_Vn-3_Vn-4*	Содержательные операнды Vn-1 и Vn
13	***Az***	*141%Ncp@Ntvad*	Терминатор входной капсулы

Любой шаблон капсулы содержит набор вспомогательных и управляющих операндов, которые имеют установленные биты ГД в БП, также как и ряд констант, значения которых заранее известны. Поэтому,

как только УП инициализирует исполнение капсулы, операнды с битами ГД = 1 начинают обрабатываться РОУ.

Содержательные операнды имеют нотацию t@sV0_[OcutShmDrse] и имеют следующие подполя, где V – поле значений данных (см. табл. 3):

- подполе размерности операнда *@s* (16/38 бит);
- подполе операции Вычислителя [*Oc*];
- подполе режима использования операнда [*Ou*] в ПС;
- подполе типа операции [*Ot*]: обычной, циклической, с инициацией перехода, считывания константы и т.д.;
- подполя кодов совпадения [*Shm*] в ПС;
- подполе маски репликации [*Dr*];
- подполе передачи операнда между секциями [*Ds*];
- подполе передачи операнда на Em-шину[*De*].

Таблица 3

Структура содержательного операнда

	Управляющая часть (УЧ)			Функциональная часть (ФЧ)								Рекуррентная ФЧ				РУЧ	Содержательная часть
00000000	T	~~~	@s	Dr	Sh	Sm	Ou	Oc	Ot	Ds	De	Sm	Ou	Oc	Ds	@s	V
00000000	11	000	0														
8	2	3	1	4	1	3	2	5	3	2	1	3	2	5	2	1	16
63...56	55...54	53...51	50	49...46	45	44...42	41...40	39...35	34...32	31...30	29	28...26	25...24	23...19	18...17	16	15...0

На рис. 4 приведена детализированная структура базового варианта РОУ (детализация его структуры, представленной на рис. 3). Кроме общесекционных устройств (Распределитель, Сборщик, Импликатор и другие) на рис. 4 представлена и структура одной из его четырех идентичных секций.

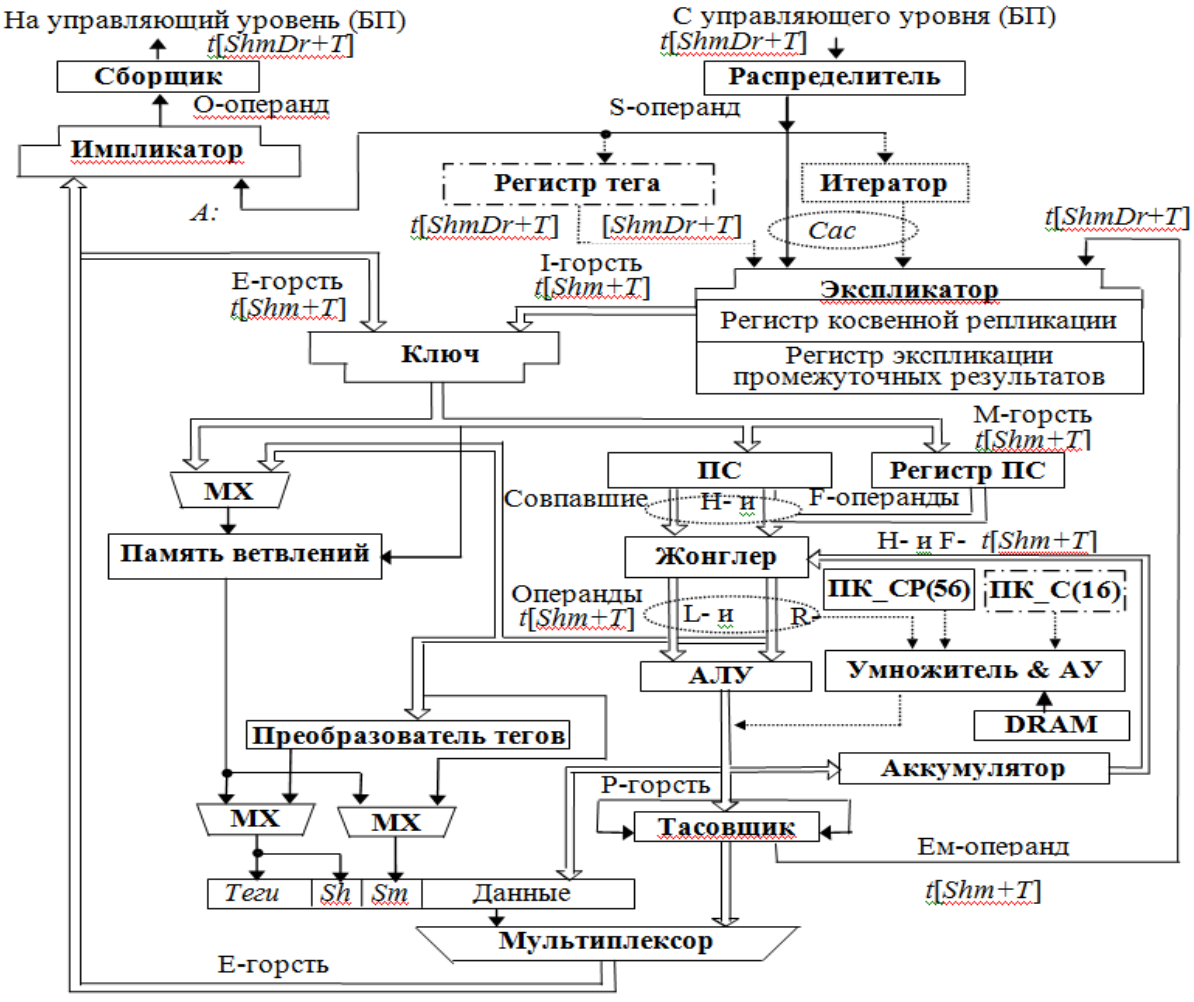

Рис. 4. Структура базового варианта РОУ

Распределитель представляет собой FIFO-буфер, получающий последовательности операндов с уста-

новленными битами ГД, которые содержатся в капсулах (БП). Распределитель является источником исходных S-операндов, которые появляются на его выходе в порядке их расположения в капсуле. Задача Распределителя – не только подача операндов на вход модуля «Экспликатор», но еще и анализ типа операнда на своём выходе (в верхней ячейке – вершине FIFO) и распределение некоторых типов вспомогательных и управляющих операндов по их месту назначения. Глубина буфера Распределителя может варьироваться.

В РОУ некоторые компоненты архитектуры имеют дело не с операндами, а с горстями. Поэтому перед обработкой операнды из капсулы должны быть преобразованы в форму горстей. Это преобразование называется *экспликацией горстей* и выполняется *Экспликатором*. Экспликатор фактически является дополнительной ступенью Распределителя, подготавливающей горсти к выдаче в пути данных секций. Экспликация является составной операцией, поскольку включает в себя *репликацию* (размножение или «векторизацию») *операнда* и собственно *экспликацию* (формирование горсти). Для репликации в функциональном подполе [*Dr*] всех исходных операндов предусмотрена явная *маска репликации*. Каждый разряд маски соответствует определенной секции. Занесение единицы в каждый из разрядов прямо адресует секцию, в которую должен быть направлен операнд. *Маска прямой репликации* инициируется ненулевым кодом репликации и прямо идентифицирует те секции, куда поступят операнды на обработку. Операнды с одной и той же маской репликации всегда направляются в одни и те же секции. В архитектуре РОУ предусмотрено несколько вариантов исполнения косвенной репликации операндов.

Подготовленная горсть (I-горсть) поступает в модуль ПС, имеющийся в каждой секции РОУ, который способен выполнять следующие функции:

- выбор и коммутация запоминаемых горстей;
- хранение операндов;
- селекция операндов, у которых совпали совпадения [*Shm*]. Пока из ПС считывается компонент пары, подполе кода операции [Oc], попавшего в регистр ПС, может быть дешифрировано, что и обеспечивает одновременность действий на шаге 1 у МПРА (см. рис. 2).

Любой модуль «Вычислитель» способен выполнять:

- арифметико-логические функции над содержательной частью операндов, используя информацию, поступающую с L и R шин: L: tV[ ] и R: tV[Oc];
- рекуррентную развертку функциональных полей операндов, используя информацию, поступающую с R-шины: [Ocut_ShmD_] в Преобразователе тегов.

Результирующие E-операнды либо запоминаются в одном из аккумуляторов Вычислителя (A, B или C), либо возвращаются в ПС по E- или Em-шинам для дальнейшего участия в вычислениях (промежуточные результаты) или для вывода на управляющий уровень с помощью Импликатора и Сборщика для выдачи окончательных результатов (НД0) или промежуточных результатов (НД1) для их повторного использования в РОУ.

Как и в стандартных цифровых сигнальных процессорах (ЦСП) Вычислитель содержит аппаратный блок MAC (Multiplication with ACcumulation), позволяющий за один такт перемножать два 16-разрядных операнда и накапливать результат умножения в одном из двух внутренних 40-разрядных регистров. Разрядности входных обрабатываемых чисел в 16 бит вполне достаточно для целого ряда применений в области речевой обработки. Для тех применений, где 16 бит недостаточно, архитектура РОС предусматривает 38 битные входные и выходные операнды.

Помимо операции умножения с накоплением блок MAC, входящий в состав РОС, способен выполнять команды арифметического, логического сдвигов и округления результатов. ВУ содержит также 16-разрядное арифметико-логическое устройство (АЛУ), выполняющее основные арифметические и все логические команды.

Каждый из перечисленных узлов и регистров ВУ представляет собой независимый аппаратный ресурс, цена простаивания которого достаточно высока. ВУ в рамках РОУ способно параллельно выполнять до двух команд за один такт, т.е. является суперскалярным.

Суперскалярным (термин впервые был использован в 1987 году) называется ВУ, которое одновременно выполняет более чем одну скалярную команду. Таким образом, реализация в ВУ суперскалярности позволила достичь параллельности на уровне операций отдельных команд.

В состав команд РОС была введена *специальная многоцикловая команда "Butt"* (Butterfly), одновременно задействующая максимальное количество функциональных узлов ВУ. Это позволило сократить до минимума число необходимых шагов для ее реализации за счет оптимального распределения ресурсов ВУ (см. [12]). При реализации такой команды параллельно задействуются все существующие аппаратные узлы ВУ, а именно – 16-разрядное АЛУ, 16-разрядный умножитель и 40-разрядный сумматор.

Для минимизации накладных расходов при реализации циклических процедур в архитектуру РОС введены следующие средства в каждой секции:

1) один 8-разрядный счетчик циклов в составе ВУ;

2) один функциональный блок – память ветвлений (ПВ) в каждом ядре;

3) четыре формата операндов: загрузчик тегов, загрузчик счетчика циклов, контроллер циклов внутренний и контроллер циклов внешний;

4) одно 2-разрядное поле типа операции.

Приведенный перечень механизмов, реализованных в РОС, позволяет повысить производительность вычислений для целого ряда ЦОС-алгоритмов.

IV. Заключение

Анализ степени соответствия принципов вычислительной парадигмы на основе потоков данных требованиям со стороны алгоритмов ЦОС показал целесообразность разработки РОС как нового поколения ЦСП, базирующегося на новой рекуррентно-динамической вычислительной парадигме.

Одним из перспективных подходов к построению РОС является его реализация в виде комбинированной двухуровневой архитектуры с ведущим фоннеймановским процессором на управляющем уровне и многими рекуррентными процессорами на операционном уровне (РОУ), непосредственно связанными между собой. На ведущий процессор возлагаются функции:

- связи многопроцессорной потоковой системы с внешним миром;
- интерфейса между стандартным программным обеспечением и многопотоковым ВП, базирующимся на потоковой парадигме;
- вычислителя для последовательных частей алгоритма;
- устройства управления для обработки исключительных ситуаций в многопроцессорной сети;
- устройства-компоновщика самодостаточных капсул для их исполнения на операционном уровне архитектуры.

Такое решение позволило подтвердить потенциальную эффективность применения потоковой парадигмы в области ЦОС [11, 12] и обеспечить совместимость с существующими вычислительными средами.

Приведенный перечень механизмов, реализованных в РОУ, позволяет повысить производительность вычислений для целого ряда алгоритмов ЦОС. Часть этих нововведений, касающихся системы команд и организации режимов функционирования Вычислителей, напрямую не связана с особенностями рекуррентной потоковой архитектуры и может быть также реализована в других ЦСП за счет максимально эффективного использования уже имеющейся аппаратуры.

Другой особенностью разрабатываемой МПРА является ориентация на самосинхронную схемотехническую базу: самосинхронизация на логическом уровне (по готовности исходных данных) хорошо сочетается с самосинхронизацией на аппаратном уровне (по готовности результатов). Поэтому, несмотря на то, что в настоящее время используется синхронный схемотехнический базис, взаимодействие между функциональными блоками и ступенями вычислительного конвейера осуществляется асинхронно.

В перспективе предполагается переход от ПЛИС-базиса с управляющим процессором NIOSII и синхронной реализацией РОУ на заказной КМОП-базис с управляющим процессором КОМДИВ и самосинхронной реализацией РОУ в виде сопроцессора. Причем для КОМДИВ уже разработаны самосинхронные сопроцессоры: 64-разрядное устройство деления и извлечения квадратного корня [13] и 64/32-разрядное устройство умножения-сложения с однократным округлением [14].

Благодарности

В заключение хотим выразить свои благодарности Морозову Н.В., Степченкову Д.Ю. и Шикунову Ю.И. за неоценимый вклад в разработку программно-аппаратной модели МПРА.

Поддержка

Исследование выполнено при частичной финансовой поддержке по подпрограмме № 4 отделения ОНИТ РАН на 2016 г. (проект 0063-2015-0016 III.3).

Литература

[1] J.Gurd, C.Kirkham, and I.Watson. The Manchester prototype dataflow computer. Commun. ACM, 28(1), Jan. 1985. P. 34-52.

[2] Arvind, and R.S. Nikhil. Executing a program on the MIT tagged token dataflow architecture. IEEE Trans. Comput., 39(3), Mar. 1990. P. 300-318.

[3] K. Hiraki, S. Sekiguchi, and T. Shimada. Status report of SIGMA-1: A data-flow supercomputer. In J.-L.Gaudiot and L.Bic, editors. Advanced Topics in Data-Flow Computing, chapter 7, Prentice Hall, Englewood Cliffs, New Jersey, 1991. P. 207-224.

[4] Бурцев В.С. Параллелизм вычислительных процессов и развитие архитектуры супер-ЭВМ. М.: Торус Пресс, 2006. 203 р.

[5] Sundararajan Sriram. Minimizing Communication and Synchronization Overhead in Multiprocessors for Digital Signal Processing. Ph.D. Dissertation, Dept. of EECS, Technical Report UCB/RL 95/90, University of California, Berkeley, CA 94720, October, 1995. 187 p.

[6] M.Chase. A Pipelined Data Flow Architecture for Digital Signal Processing: The NEC μPD7281. IEEE Workshop on Signal Processing, November 1984.

[7] Iiro Hartimo. DFSP: A Data Flow Signal Processor. IEEE Transactions on Computers, v. C-35, N 1, January 1986. P. 23-33.

[8] K. Kronlof, J. Skytta, 0. Simula, I. Hartimo. Simulation of a digital signal processing architecture based on the data flow principle. Proc. ISCAS'82, Rome, Italy, May 10-12, 1982. P. 1053-1056.

[9] Arvind. The Evolution of Dataflow Architecture from Static Dataflow to P-RISC. Proc. of Workshop on Massive Parallelism: Hardware, Programming and Application, Amalfi, Italy, October 1989. Academic Press, 1990.

[10] Волчек В.Н., Степченков Ю.А., Петрухин В.С., Прокофьев А.А. Цифровой сигнальный процессор с нетрадиционной рекуррентной потоковой архитектурой // Проблемы разработки перспективных микро- и наноэлектронных систем - 2010. Сборник трудов / под общ. ред. академика А.Л.Стемпковского. М.:ИППМ РАН, 2010. С. 412-417.

[11] Yu. Shikunov, D. Khilko, Yu. Stepchenkov. Hardware and Software Modelling and Testing of Non-Conventional Data-Flow Architecture // Proceedings of the 2016 IEEE North West Russia Section Young Researchers in Electrical and Electronic Engineering Conference (ElConRusNW), 2016. P. 360-364.

[12] Степченков Ю.А., Волчек В.Н., Петрухин В.С., Прокофьев А.А. Механизмы обеспечения поддержки алгоритмов цифровой обработки речевых сигналов в рекуррентном обработчике сигналов // Системы и средства информатики – М.: ТОРУС ПРЕСС. Вып.20. № 1, 2010. С. 31-47.

[13] Stepchenkov Y., Diachenko Y., Zakharov V., Rogdestvenski Y., Morozov N., Stepchenkov D. Quasi-Delay-Insensitive Computing Device: Methodological Aspects and Practical Implementation // PATMOS'2009: Proceedings of the International Workshop on power and timing modeling, optimization and simulation. – Delft, The Netherlands, Springer 2010. P. 276–285.

[14] Stepchenkov Yu., Zakharov V., Rogdestvenski Yu., Diachenko Yu., Morozov N., Stepchenkov D. Speed-Independent Floating Point Coprocessor // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2015), Batumi, Georgia. 2015. P. 111-114.

Recurrent data-flow architecture: features and realization problems

Yu. A. Stepchenkov, Yu. G. Diachenko, D. V. Khilko, V.S. Petrukhin

The Institute of Informatics Problems, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences

ystepchenkov@ipiran.ru, diaura@mail.ru, dhilko@yadenx.ru, cokrat2@rambler.ru

***Keywords* — data-flow architecture, recurrence, digital signal processing, match memory, superscalarity.**

EXTENDED ABSTRACT

Results of development of the multi-core recurrent data-flow architecture (MRDA) focused on effective implementation of digital signal processing (DSP) algorithms are presented. Principal differences between MRDA and existing computer architectures are shown. Such differences make it possible to process the instructions in almost half the normal time using singular self-sufficient recurrently represented data-flow. One of the perspective approaches to building MRDA is presented in a form of a hybrid two-level architecture utilizing von Neumann control unit (NIOSII) and multi-core recurrent operational unit. Success of such solution has proven the effectiveness of data-flow paradigm in DSP and compatibility with existing computing environments.

Multiple non-recurrent mechanisms are shown to improve the performance of multiple DSP algorithms. Some of the mechanisms could be used in traditional DSP systems.

Another feature of MRDA is a perspective of self-timed implementation derived from the deep coherence between self-timed logic (data is ready) and self-timed hardware (results are ready). Currently, despite synchronous design, the interactions between functional blocks and instruction pipeline stages are asynchronous.

The future transition from synchronous FPGA-basis with NIOSII control unit towards custom-made CMOS-basis with KOMDIV control unit and a self-timed recurrent operational device in a form of a coprocessor is possible. Self-timed 64-bit division and square root unit as well as 64/32-bit MAC with rounding unit are already developed in a form of coprocessors.

SUPPORT

The study was partially supported by the 2016's subprogram no. 4 of ONIT RAS department. (Project 0063-2015-0016 III.3).

REFERENCES

[1] J. Gurd, C. Kirkham, and I. Watson. The Manchester prototype dataflow computer. Commun. ACM, 28(1), Jan. 1985. P. 34-52.

[2] Arvind, and R.S. Nikhil. Executing a program on the MIT tagged token dataflow architecture. IEEE Trans. Comput., 39(3), Mar. 1990. P. 300-318.

[3] K. Hiraki, S. Sekiguchi, and T. Shimada. Status report of SIGMA-1: A data-flow supercomputer. In J.-L.Gaudiot and L.Bic, editors. Advanced Topics in Data-Flow Computing, chapter 7, Prentice Hall, Englewood Cliffs, New Jersey, 1991. P. 207-224.

[4] Burcev V.S. Overlapping of computing processes and development of architecture of the supercomputer. M.: Torus Press, 2006. 203 p.

[5] Sundararajan Sriram. Minimizing Communication and Synchronization Overhead in Multiprocessors for Digital Signal Processing. Ph.D. Dissertation, Dept. of EECS, Technical Report UCB/RL 95/90, University of California, Berkeley, CA 94720, October, 1995.

[6] M. Chase. A Pipelined Data Flow Architecture for Digital Signal Processing: The NEC μPD7281. IEEE Workshop on Signal Processing, November 1984.

[7] Iiro Hartimo. DFSP: A Data Flow Signal Processor. IEEE Transactions on Computers, v. C-35, No. 1, January 1986. P. 23-33.

[8] K. Kronlof, J. Skytta, 0. Simula, I. Hartimo. Simulation of a digital signal processing architecture based on the data flow principle. Proc. ISCAS'82, Rome, Italy, May 10-12, 1982. P. 1053-1056.

[9] Arvind. The Evolution of Dataflow Architecture from Static Dataflow to P-RISC. Proc. of Workshop on Massive Parallelism: Hardware, Programming and Application, Amalfi, Italy, October 1989. Academic Press, 1990.

[10] Volchek V.N., Stepchenkov Yu.A., Petrukhin V.S., Prokofyev A.A., Zelenov R.A. Digital Signal Processor With Non-Conventional Recurrent Data-Flow Architecture // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2010. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2010. P. 412-417 (in Russian).

[11] Yu. Shikunov, D. Khilko, Yu. Stepchenkov Hardware and Software Modelling and Testing of Non-Conventional Data-Flow Architecture // Proceedings of the 2016 IEEE North West Russia Section Young Researchers in Electrical and Electronic Engineering Conference (ElConRusNW), 2016. P. 360-364.

[12] Yu. Stepchenkov, V. Volchek, V. Petrukhin, A. Prokofyev. Hardwere maintenance for digital processing of speech signals in the recurrent dataflow processor // Systems and means of informatics – TORUS PRESS, Moscow, 2010. P. 31-47. (in Russian).

[13] Stepchenkov Y., Diachenko Y., Zakharov V., Rogdestvenski Y., Morozov N., Stepchenkov D. Quasi-Delay-Insensitive Computing Device: Methodological Aspects and Practical Implementation // PATMOS'2009: Proceedings of the International Workshop on power and timing modeling, optimization and simulation. – Delft, The Netherlands, Springer 2010. P. 276–285.

[14] Stepchenkov Yu., Zakharov V., Rogdestvenski Yu., Diachenko Yu., Morozov N., Stepchenkov D. Speed-Independent Floating Point Coprocessor // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2015), Batumi, Georgia. 2015. P. 111-114.