Предварительная оценка производительности алгоритмических ядер цифровых фильтров для рекуррентной потоковой архитектуры

Юрий А. Степченков[1], Дмитрий В. Хилько[2], Юрий И. Шикунов[3], Георгий А. Орлов[4]
Институт проблем информатики
Федеральный исследовательский центр «Информатика и управление» Российской академии наук
Москва, Россия
[1]YStepchenkov@ipiran.ru, [2]dhilko@yandex.ru, [3]yishikunov@gmail.com, [4]Orlov.jaja@gmail.com

***Аннотация*—В статье рассматриваются результаты предварительной оценки производительности потоковой рекуррентной архитектуры на подмножестве основных задач цифровой обработки сигналов. Оцениваются различные варианты реализации свертки векторов, КИХ-фильтров, БИХ-фильтров, адаптивного фильтра и 256-Point-In-Place FFT. Реализация перечисленных алгоритмов осуществлена на основе TMS320C55x DSP Library. Полученные предварительные результаты показали, что уровень производительности рекуррентной архитектуры, основанной на потоковых принципах, не уступает TMS320C55x, основанному на принципах фон Неймана, по количеству циклов вычислений. Также представлены предложения по развитию архитектуры на основе полученных результатов.**

Keywords—бенчмаркинг; цифровая обработка сигналов; алгоритмические ядра

I. Введение

Потоковые вычислительные архитектуры обладают высоким потенциалом производительности на ряде предметных областей. Тем не менее, потоковая модель вычисления не получила широкого распространения. Мы разрабатываем потоковую гибридную архитектуру рекуррентного обработчика сигналов (ГАРОС). Апробация архитектуры осуществляется в области цифровой обработки сигналов (ЦОС), а демонстрационная задача – распознавание изолированных слов (РИС) [1].

Результаты испытаний архитектуры на программной и аппаратной моделях для выбранной задачи показали высокий потенциал ее эффективности. Поэтому на основе аппаратной модели был разработан макетный образец архитектуры в ПЛИС технологии с использованием Cyclone V GT Development Kit. В статье [2] приводятся результаты испытаний полученного макетного образца. Таким образом, нам удалось показать работоспособность вычислительного устройства, основанного на данной архитектуре.

Основным способом оценки производительности, рассмотренным в работе [2], является задача РИС. Сравнение осуществлялось с показателями быстродействия реализации данной задачи на микроконтроллере. Однако, данная задача и методика измерения не предоставляют исчерпывающей информации, чтобы можно было сделать вывод об эффективности архитектуры для ЦОС применений. Поэтому необходимо более полное и доказательное тестирование.

Согласно определению, данному в [3], оценкой производительности называется процесс, анализирующий поведение системы при конкретных исходных условиях. Авторы в [3] определяют такие категории условий: “заданное поведение, заданная нагрузка, или заданный набор входных данных.”. Организованное таким образом измерение производительности позволяет с самых ранних этапов разработки выбрать наиболее подходящие архитектурные решения, которые позволят добиться требуемых параметров производительности или обнаружить возможные улучшения уже существующего дизайна. Одним из наиболее распространенных методов измерения производительности вычислительных устройств является бенчмаркинг – испытание устройств на заданном комплекте задач с фиксированным набором метрик. Поэтому было принято решение провести benchmarking ГАРОС как для доказательства жизнеспособности, так и для поиск возможных проблем архитектуры с целью ее совершенствования.

Существует множество различных способов бенчмаркинга. В статье [4] приводится описание наиболее востребованных подходов к бенчмаркингу ЦОС процессоров, а также их преимуществ и недостатков. К ним относятся: синтетические тесты, алгоритмические ядра, прикладные тесты, гибридные подходы и пользовательские оценки. Мы провели анализ перечисленных подходов и пришли к выводу, что наиболее подходящим для ГАРОС является подход с использованием алгоритмических ядер, в силу того, что «Алгоритмическое ядро - это небольшая программа, взятая из реального приложения или набора приложений. Алгоритмические ядра используют правило 80/20, которое гласит, что 20% данного фрагмента кода будут составлять

Работа выполнена при поддержке гранта Российского научного фонда (проект № 19-11-00334).

978-0-7381-4275-3/21/$31.00 ©2021 IEEE

80% рабочей нагрузки, связанной с этим кодом.» [4]. Кроме того, недостатки алгоритмических ядер являются несущественными в рамках поставленной задачи для ГАРОС.

В статье [5] приводится сравнение двух основных поставщиков бенчмарков: Berkeley Design Technology, Inc. (BDTI) и Embedded Microprocessor Benchmark Consortium (EEMBC). ПРи этом набор алгоритмических ядер от BDTI сосредоточен на измерении производительности ЦОС процессоров, в то время как EEMBC - для аппаратного и программного обеспечения, используемого для автономного вождения, мобильной обработки изображений, Интернета вещей, мобильных устройств и многих других приложений. Также в этой работе автор делает вывод, что BDTI Benchmarks лучше подходят именно для ЦОС процессоров, в то время как EEMBC – лидирует в других areas. Поэтому мы приняли решение реализовать BDTI Benchmarks для оценки производительности ГАРОС.

В качестве эталона для сравнения производительности мы выбрали семейство ЦОС микропроцессоров TMS320C55x. Целью исследований в рамках настоящей статьи является оценка производительности ГАРОС на базе ограниченного множества алгоритмических ядер BDTI и постановка задачи развития архитектуры для достижения требуемых параметров ее производительности.

II. Алгоритмические ядра ЦОС

A. Методика оценки производительности

В книге [3] рассматривается продукция компании BDTI, как одного из лидеров бенчмаркинга DSP процессоров. В 1994 году BDTI представила свои тесты DSP, которые получили название «BDTI Benchmarks». BDTI Benchmarks включает 12 ядер алгоритмов, которые представляют основные операции ЦОС, используемые в общих приложениях ЦОС. В таблице 1 (таблица 10.7 из [3]) приводится таблица алгоритмических ядер, измеряемых в BDTI Benchmarks.

Алгоритмическими ядрами называются функции, которые представляют собой основные строительные блоки большинства задач обработки сигналов. Эти ядра являются наиболее ресурсоемкими частями ЦОС задач. Подразумевается, что эти ядра высоко оптимизированы и разрабатываются вручную для каждого конкретного ЦОС процессора.

Поэтому было принято решение разработать и оценить производительность для РОУ собственного набора алгоритмических ядер, основанного на аналогичных алгоритмах, т.к. «BDTI Benchmarks» является закрытым программным продуктом. В дальнейшем планируется использовать данные разработки для создания собственной низкоуровневой библиотеки ЦОС алгоритмов.

В качестве эталона для реализации и сравнения мы используем публичную документацию ЦОС микропроцессоров TMS320C55x (семейства C55x) компании Texas Instruments (TI), распространяемых в настоящее время. Причинами данного выбора являются:

- Продукты компании TI занимают лидирующие позиции на рынке ЦОС микропроцессоров;
- Библиотека TMS320C55x ЦОС хорошо документирована и содержит описание, реализацию и оценки производительности по количеству циклов различных алгоритмов ЦОС;
- Указанная библиотека в полном объеме реализует BDTI Benchmarks.

Мы проанализировали библиотеку ЦОС TMS320C55x [6] и выбрали необходимые функции, реализующие алгоритмические ядра BDTI Benchmarks. Оказалось, что указанная библиотека содержит несколько версий реализации этих ядер, ввиду специфики аппаратной реализации линейки C55x. Кроме того, в C55x ЦОС библиотеке четко прописано следующее: « Предполагается, что все данные находятся во встроенной оперативной памяти с двусторонним доступом (предоставленный командный файл компоновщика отражает эти условия)». Поэтому в целях полноты сравнения были реализованы соответствующие версии для РОС, оцениваемые с аналогичными предположениями. Далее приводятся описания и результаты оценки производительности для каждого из алгоритмических ядер.

Таблица I Алгоритмические ядра BDTI (таблица 10.7 из [3])

Функция	**Описание функции**	**Пример использования**
Real Block FIR	Фильтр с конечной импульсной характеристикой (КИХ), который выполняется на блоке действительных данных.	Обработка речи (например G.728 кодирование речи).
Complex Block FIR	КИХ-фильтр, который выполняется на блоке комплексных данных.	Выравнивание модемного канала.
Real Single-Sample FIR	КИХ-фильтр, который выполняется на одиночном действительном отсчете.	Обработка речи, общая фильтрация.
LMS Adaptive FIR	Адаптивный фильтр наименьших квадратов; работает с одиночным действительным отсчетом.	Выравнивание каналов, сервоуправление, кодирование с линейным предсказанием.
IIR	Фильтр с бесконечной импульсной характеристикой (БИХ), работающий с одиночным действительным отсчетом.	Обработка звука, общая фильтрация.
Vector Dot Product	Сумма поточечного умножения двух векторов.	Свертка, корреляция, умножение матриц, многомерная обработка сигналов.
Vector Add	Поточечное сложение двух векторов, в результате чего получается третий вектор.	Графика, объединение звуковых сигналов или изображений.

Функция	Описание функции	Пример использования
Vector Maximum	Поиск значения и расположения максимального значения в векторе.	Кодирование с контролем ошибок, алгоритмы с блочной плавающей точкой.
Viterbi Decoder	Декодирование блока битов, который был закодирован сверточным кодированием.	Кодирование с контролем ошибок.
Control	Последовательность управляющих операций (тест, ветвление, push, pop и манипулирование битами).	Практически все приложения ЦОС содержат управляющий код.
256-Point In-Place FFT	Быстрое преобразование Фурье преобразует сигнал временной области в частотную.	Радар, сонар, сжатие звука MPEG, спектральный анализ.
Bit Unpack	Распаковывает данные переменной длины из битового потока.	Декомпрессия звука, обработка протокола.

B. Описание алгоритмических ядер

В рамках данной статьи не рассматриваются аспекты реализации алгоритмических ядер “Управление”, “Распаковка битов” и “Декодер Витерби”. Как и любой другой ЦОС процессор, РОУ обеспечивает набор управляющих инструкций, за исключением инструкций для манипуляции битами. Данный недостаток связан с исчерпанностью резервов текущей реализации системы команд. По этой причине реализовать “Распаковка битов” не представляется возможным. А реализация “Декодер Витерби” является нетривиальной задачей, связанной с необходимостью совершенствования архитектуры РОУ. Поэтому в данной статье дана только предварительная оценка реализации данных алгоритмических ядер.

1) Real block FIR

Данное алгоритмическое ядро реализует блочный фильтр с конечной импульсной характеристикой, передаточная функция которого в общем виде задается формулой (1).

$$r[j] = \sum_{k=0}^{nh-1} h[k]x[j-k], 0 \le j \le nx \quad (1)$$

Здесь: *x* – блок входных отсчетов, *h* – блок коэффициентов фильтра, *nh* – количество коэффициентов фильтра, *nx* – количество отсчетов.

Особенностью блочного фильтра является то, что он осуществляет обработку сразу целого блока данных и поэтому не может быть использован в реальном времени (т.к. имеет непостоянную скорость получения фильтрованных отсчетов).

2) Real Single-Sample FIR

Основным отличием данного фильтра от блочного является постоянная скорость получения выходных отсчетов. На каждый один входной отсчет приходится один выходной. Таким образом, данный фильтр может быть использован для обработки отсчетов, поступающих в реальном масштабе времени. Функция фильтрации задается также формулой (1) с тем лишь отличием, что nx=1.

3) Complex Block FIR

Рассматриваемое алгоритмическое ядро является блочной версией КИХ-фильтра (1), в котором и входные отсчеты, и коэффициенты являются комплексными числами. Для вычисления данного фильтра необходимо произвести преобразования уравнения (1). Полученные уравнения (2)-(3) имеют следующий вид:

$$r_r(j) = \sum_{k=0}^{nh-1}(h_r(k) * x_r(j-k) - h_i(k) * x_i(j-k)) \quad (2)$$

$$r_i(j) = \sum_{k=0}^{nh-1}(h_i(k) * x_r(j-k) + h_r(k) * x_i(j-k)) \quad (3)$$

4) LMS Adaptive FIR

Адаптивный фильтр используется в случае фильтрации зашумленного сигнала. При этом его коэффициенты пересчитываются от шага к шагу, что позволяет более эффективно удалять шумы и приближать выходной сигнал к ожидаемому. В библиотеке ЦОС C55x реализован адаптивный алгоритм наименьших квадратов с задержкой. Данный фильтр включает в себя этап КИХ, определяемый формулой (1), а адаптация описывается формулами (4) и (5):

$$e[i] = des[i] - r[i] \quad (4)$$

$$h_k[i+1] = h_k[i] + 2 * \mu * e[i] * x[i-k] \quad (5)$$

Здесь: *e[i]* – рассчитываемое значение ошибки, а μ - значение скорости спуска к минимуму ошибки.

5) Real Single-Sample IIR

Высококачественные частотные КИХ фильтры, как правило, требуют большой ширины окна (количества коэффициентов). Альтернативным решением является использование каскадных фильтров с бесконечной импульсной характеристикой, которые являются рекурсивными и позволяют значительно сократить количество коэффициентов.

В рамках библиотеки ЦОС C55x реализованы две основные формы записи одной секции каскадных БИХ фильтров, называемой также биквадами : Прямая форма I и прямая форма II. Формула (6) определяет БИХ формы I, а формулы (7) и (8) – БИХ формы II соответственно.

$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] - a_1 * y[n-1] - a_2 * y[n-2] \quad (6)$$

$$d[n] = x[n] - a_1 * d[n-1] - a_2 * d[n-2] \quad (7)$$

$$y[n] = b_0 * d[n] + b_1 * d[n-1] + b_2 * d[n-2] \quad (8)$$

Здесь a_1, a_2, b_0, b_1, b_2 – коэффициенты биквада. Количество каскадов (биквадов) задается параметром *nbiq*.

6) Vector Dot Product

Скалярное произведение векторов вычисляется как сумма покоординатных произведений. Очевидно, что вектора должны иметь одинаковую длину.

7) Vector Add

Сумма векторов это вектор, координаты которого вычисляются как суммы соответствующих координат исходных векторов. Очевидно, что вектора должны иметь одинаковую длину.

8) Viterbi Decoder

За основу реализации декодера Витерби взяты рекомендации компании TI [7]. В ГАРОС можно реализовать только часть алгоритма – вычисление метрики.

9) 256-point In-Place FFT

Данное алгоритмическое ядро является реализацией алгоритма «radix-2 с прореживанием по времени БПФ» с использованием типовой операции «бабочка» [8]. Кроме того, полученные результаты записываются на место исходных (in-place реализация).

III. Реализация алгоритмических ядер

A. *Описание реализации алгоритмических ядер*

Особенности аппаратной реализации семейства микропроцессоров C55x допускают различные варианты реализации kernel benchmarks. Поэтому в библиотеке ЦОС C55x определено несколько версий функций для каждого из ядер. С учетом этого мы также реализовали различные версии алгоритмов для более точной оценки производительности.

1) Real block FIR

В ЦОС Lib C55x определено две версии данного ядра. Первая версия использует 1 MAC блок. Для ее реализации в ГАРОС используется: *одна* секция (1 MAC блок), количество итераций равно *nx*, не требуется специальной рутины на управляющем уровне.

Вторая версия использует 2 MAC блока. Для ее реализации в ГАРОС используется: *две* секции (2 MAC блока), количество итераций равно *nx*, не требуется специальных рутин на управляющем уровне.

2) Real Single-Sample FIR

Реализация данного фильтра в библиотеке ЦОС C55x использует 2 MAC блока. Для его реализации в ГАРОС используется: *две* секции (2 MAC блока), количество итераций равно *1*, требуется специальная рутина на управляющем уровне для подгрузки отсчетов. Кроме того, мы разработали две версии данного алгоритма: реального времени и псевдо реального времени (два выходных отсчета каждые два периода семплирования).

3) Complex Block FIR

Реализация данного фильтра в библиотеке ЦОС C55x использует 2 MAC блока. Для его реализации в ГАРОС используется: *две* секции (2 MAC блока), количество итераций равно *nx*, не требуется специальной рутины на управляющем уровне, используется подгружаемая память констант для коэффициентов фильтра.

4) LMS Adaptive FIR

Реализация данного фильтра в библиотеке ЦОС C55x использует 2 MAC блока, а также является фильтром с задержкой. Для его реализации в ГАРОС используется: *две* секции (2 MAC блока), количество итераций равно *nx*, не требуется специальной рутины на управляющем уровне, но реализуется фильтр без задержки.

5) Real Single-Sample IIR

В библиотеке ЦОС C55x определено четыре версий данного ядра, использующих 2 MAC блока. Первая версия реализует фильтр двойной точности в форме II (т.е. все отсчеты и коэффициенты это 32-разрядные числа). Вторая версия реализует фильтр с 4 коэффициентами биквада в форме II. Третья версия реализует фильтр с 5 коэффициентами биквада в форме II. Четвертая версия реализует фильтр с 5 коэффициентами биквада в форме I.

Для реализации всех версий в ГАРОС используется: *две* секции (2 MAC блока), количество итераций равно *1*, требуется специальная рутина на управляющем уровне для подгрузки отсчетов.

6) Vector Dot Product

Реализация является тривиальной.

7) Vector Add

Реализация является тривиальной.

8) 256-point In-Place FFT

Рассматривается реализация без аппаратного ускорителя для C55x. Используется масштабирование данных для избегания переполнения. Согласно бибилотеке ЦОС C55x данная версия обрабатывает одну операцию «бабочка» за 5 циклов. Реализация в ГАРОС использует специальную инструкцию BUTT, которая вычисляет «бабочку» за 4 цикла.

B. *Анализ результатов*

Сравнительные результаты оценки производительности приводятся в таблице II для каждой из версии реализации алгоритмических ядер.

Таблица II Результаты бенчмаркинга ГАРОС

Функция	**Оценка количества циклов**	
	C55x	***ГАРОС***
Block FIR 1 MAC	C[a]: nx * (2 + nh) O[b]: 25	C: nx * (1 + nh) O: 12
Block FIR 2 MAC	C: nx/2 * (6 + nh) O: 25	C: nx/2 * (1 + nh) O: 12
Single Sample FIR 1 MAC	C: (1 + nh) O: 44	C: (1 + nh) O:15
Single Sample FIR 2 MAC	C: (1 + nh/2) O: 24	C: (3 + nh/2) O: 15
Quasi-realtime Single Sample FIR 2 MAC	-	C: (1 + nh/2) O: 24
Complex Block FIR	C: nx * [8 + 2*(nh-2)] O: 51	C: nx * (1 + 2*nh) O: 14
LMS Adaptive FIR	C: nx * (5 + 2*nh) O: 26	C: nx * (4 + 3*nh) O: 12
IIR Double Precision Form II	C: nx * (7 + 31*nbiq[c]) O: 77	C: nx * (4 + 21 * nbiq) O: 12
IIR 4 Coefficient Form II	C: nx * (2 + 3*nbiq) O: 44	C: nx * (2 + 5*nbiq) O: 12
IIR 5 Coefficient	C: nx * (5 + 5*nbiq) O: 60	C: nx * (2 + 6*nbiq) O: 12

Функция	Оценка количества циклов	
	C55x	*ГАРОС*
Form II		
IIR 5 Coefficient Form I	C: nx * (5 + 8*nbiq) O: 68	C: nx * (1 + 7*nbiq) O: 12
Vector Dot Product	C: nx + 1 O: 44	C: nx + 1 O: 15
Vector Add	C: nx * 3 O: 23	C: nx * 2 O: 14
Vector Maximum	C: nx * 3 O: 8	C: nx * 5 O: 12
Viterbi Decoder[d]	C: Frame * 34 O: 121	C: Frame * 109 O: n/a
256-Point In-Place FFT	C: 5 * N/2 * log2N	C: 4 * N/2 * log2N

[a] *Core - количество циклов, необходимых для выполнения алгоритмического ядра*

[b] *Overhead - накладные расходы, не зависящие от входных параметров алгоритма*

[c] *Количество биквадов*

[d] *Используется GSM Half Rate Convolutional Encoder (R=1/2, K=5, Frame=189), Вычисляется метрика*

Как видно из таблицы II большинство алгоритмических ядер имеют хорошие оценки производительности, практически не уступающие аналогичным реализациям для C55x. Однако несколько алгоритмов имеет существенно более низкую производительность. Мы провели детальный анализ архитектуры микропроцессора C55x и обнаружили ряд интересных архитектурных решений, обеспечивающих столь высокую его производительность.

На основе результатов данного анализа мы смогли определить набор задач, решение которых позволит добиться требуемого уровня производительности:

- Повышение степени использования суперскалярности ГАРОС (параллелизм уровня инструкций);
- Исследование возможности применения VLIW технологии для ГАРОС по аналогии с C55x, который является VLIW процессором;
- Переработка и значительное расширение системы команд для обеспечения адекватного наполнения VLIW-инструкций;
- Увеличение мощности регистрового файла в вычислительном блоке для обеспечения VLIW-инструкций необходимым количеством данных;
- Внедрение механизмов быстрой загрузки данных в регистровые файлы вычислительных блоков минуя необходимые этапы образования пар;
- Повышение гибкости и функциональности памяти констант.

IV. Заключение

Результаты бенчмаркинга ГАРОС позволяют сделать вывод, что архитектура является жизнеспособной и обладает хорошим потенциалом производительности в классе задач ЦОС. В частности, большинство алгоритмических ядер показали идентичный уровень производительности, как и продукт компании TI, лидирующей на рынке ЦОС процессоров.

Тем не менее, бенчмаркинг выявил ряд архитектурных проблем, из-за которых определенные задачи решаются неэффективно. Поэтому мы более глубоко и подробно изучили архитектуры современных ЦОС процессоров и методы реализации типов задач из этого класса. В результате был разработан набор предложений по развитию ГАРОС, внедрение которых позволит достигнуть желаемого уровня производительности.

Таким образом, в результате данного исследования нам удалось достичь поставленных целей исследования. Дальнейшее развитие ГАРОС мы видим в разработке на основе алгоритмических ядер и библиотеки ЦОСb C55x собственной высоко оптимизированной ЦОС библиотеки. Эту библиотеку следует также снабдить средствами высокоуровневого описания на языке Си. Успешная разработка данной ЦОС библиотеки обеспечит конкурентоспособность ГАРОС на рынке отечественных ЦОС решений.

Благодарности

В заключении мы хотим поблагодарить Дьяченко Ю.Г. и Рождественского Ю.В. за существенный вклад в разработку ГАРОС.

REFERENCES

[1] Yu. Stepchenkov, D. Khilko, Yu. Diachenko, Yu. Shikunov and D. Shikunov, "Testing of software and hardware simulations of dataflow recurrent digital signal processor," 2016 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, 2016, pp. 168-171, doi: 10.1109/EWDTS.2016.7807672.

[2] Y. Stepchenkov, Y. Shikunov, N. Morozov, G. Orlov and D. Khilko, "Hybrid Multi-Core Recurrent Architecture Approbation on FPGA," 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg and Moscow, Russia, 2019, pp. 1705-1708, doi: 10.1109/EIConRus.2019.8657140.

[3] Lavagno L., Martin G., Markov I.L., Scheffer L.K. Electronic Design Automation for IC System Design, Verification, and Testing. CRC Press; 2nd edition, 2016 - 664 p.

[4] Kenton Williston “Benchmarking basics, part 1: Choosing and using benchmarks”, Jul 25 2008. Available at: https://www.eetimes.com/benchmarking-basics-part-1-choosing-and-using-benchmarks/ (Accessed 30 November 2020).

[5] Kenton Williston “Benchmarking basics, part 2: BDTI and EEMBC reviewed”, Jul 2 2008. Available at: https://www.edn.com/ benchmarking-basics-part-2-bdti-and-eembc-reviewed/ (Accessed 30 November 2020).

[6] TMS320C55x DSP Library Programmer’s Reference, Texas Instruments. Rev May 2013. Available at: https://www.ti.com/lit/pdf/spru422. / (Accessed 30 November 2020).

[7] Henry Hendrix, “Viterbi Decoding Techniques for the TMS320C55xDSP Generation”, Texas Instruments. April 2009. Available at: https://www.ti.com/lit/an/spra776a/spra776a.pdf. / (Accessed 30 November 2020).

[8] Eleanor Chu, Alan George. Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms, CRC Press LLC, 2000. - 308 p.