FACT EXTRACTION FROM NATURAL LANGUAGE TEXTS WITH CONCEPTUAL GRAPH MODELS

M.Yu. Bogatyrev

Applications of methods of conceptual modeling in the fact extraction technology from textual data are considered. Textual data is presented as non-structured natural language texts, forming a text corpus. Conceptual graphs and concept lattices are used as conceptual models. The use of conceptual graphs allows to solve effectively the problems of named entity recognition and relations extraction. These solutions then applied for building concept lattices which serve as a data source in the fact extraction system. Extraction of facts is doing by processing user requests and finding the corresponding concepts in the concept lattice..

Key words: conceptual modeling, conceptual graphs, conceptual lattices, formal concept analysis.

Bogatyrev Mikhail Yurievich, doctor of technical sciences, professor, okkambo@mail.ru, Russia, Tula, Tula State University

УДК 621.391:519.72

РАЗРАБОТКА МОДЕЛИ СИСТЕМЫ ПЕРЕДАЧИ ДАННЫХ С МНОГОПОРОГОВЫМ ДЕКОДЕРОМ САМООРТОГОНАЛЬНЫХ КОДОВ С ПРИМЕНЕНИЕМ ТЕХНОЛОГИИ OPENCL

Д.С. Демидов, Г.В. Овечкин

Выполнен анализ особенностей моделирования систем передачи данных с многопороговым декодированием МПД самоортогональных кодов. Показано, что для ускорения процесса моделирования можно использовать вычислительные ресурсы GPU. Разработана компьютерная модель системы передачи данных с МПД с использованием GPU, рассмотрены возможности ее ускорения. Показано, что применение предложенной модели позволяет увеличить скорость работы модели до 40 раз по сравнению с аналогичной моделью на CPU.

Ключевые слова: системы передачи данных, помехоустойчивое кодирование, многопороговое декодирование, компьютерное моделирование, OpenCL, GPU, CPU.

Быстрый рост объемов обработки данных, развитие цифровых систем вещания и вычислительных сетей предъявляют высокие требования к минимизации ошибок в используемых цифровых данных. Поэтому одной из важнейших задач является обеспечение высокой достоверности передачи данных [1]. Решением задачи обеспечения высокой достоверности передачи данных занимается помехоустойчивое кодирование.

Одно из наиболее эффективных решений проблемы повышения достоверности передачи данных при небольшой сложности реализации основано на использовании многопороговых декодеров (МПД), в основе работы которых лежит итеративное декодирование самоортогональных кодов (СОК) [2]. Известно, что МПД с линейной от длины кода сложностью реализации позволяют выполнять близкое к оптимальному декодирование правильно выбранных кодов.

Для оценки эффективности помехоустойчивых кодов и методов их декодирования при большом уровне шума требуется использовать моделирование, поскольку аналитическая оценка вероятности ошибки в таких условиях является обычно очень неточной. Особенностью современных систем передачи и хранения данных является низкая целевая вероятность ошибки. Например, в ряде таких систем необходимо обеспечить вероятность ошибки менее 10^{-11}. Для оценки подобной вероятности ошибки с помощью моделирования нужно выполнить передачу как минимум 10^{12} битов. Моделирование передачи такого объема данных требует недопустимо больших временных затрат. Это делает актуальной задачу разработки и реализации модели системы передачи данных, позволяющей выполнять такие объемы эксперимента за приемлемое время. В данной работе для решения поставленной задачи предлагается использовать вычислительные ресурсы графических процессоров, для работы с которыми применяется язык OpenCL.

Отметим что при построения эффективной модели необходимо оценить сложность реализации каждого узла системы передачи данных как с точки зрения количества элементарных арифметических операций, требующихся для передачи одного информационного бита, так и по числу обращений к памяти.

Общая схема передачи данных. В процессе создания модели необходимо выполнить анализ общей схемы системы передачи данных, представленной на рис.1, и оценить вычислительные затраты, приходящиеся на каждый узел системы.

Представленная схема состоит из семи узлов.

Источник данных генерирует данные, предназначенные для передачи. В общем случае он формирует 0 и 1 с равной вероятностью.

Кодер канала кодирует передаваемые данные путем добавления к ним некоторой избыточности. В случае оценки эффективности МПД используется кодер блокового СОК, пример схемы которого для СОК, заданного полиномом g=(0, 1, 4, 6), представлен на рис. 2.

Модулятор адаптирует данные для передачи по каналу, выполняя отображение набора входных бит в некоторый аналоговый сигнал, зависящий от вида модуляции. В данной работе рассматривается только простейший случай двоичной фазовой модуляции (BPSK).

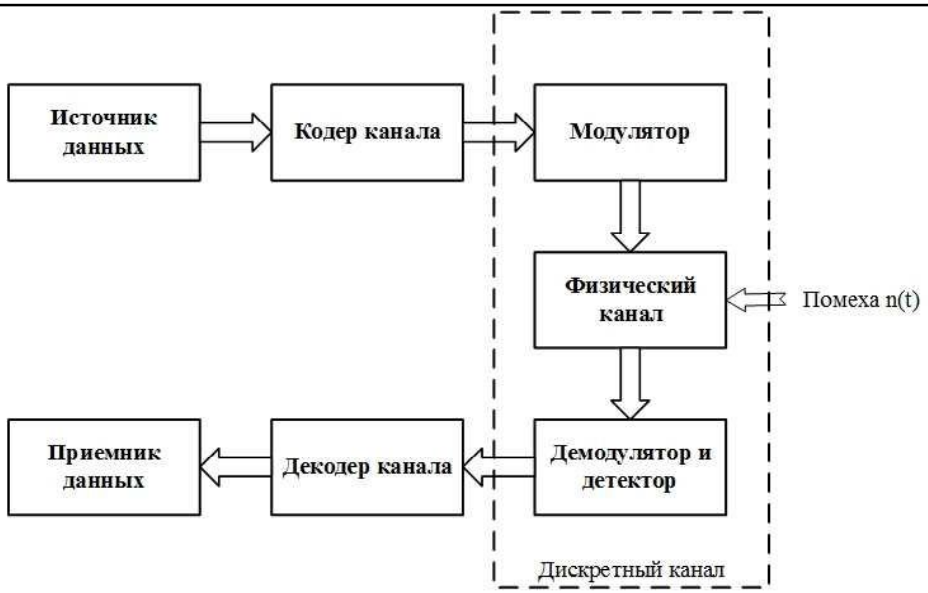

Рис. 1. Общая схема системы передачи данных

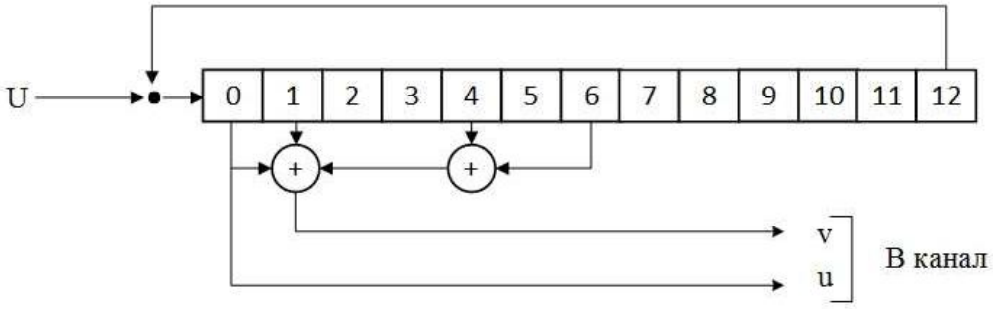

Рис. 2. Схема кодера блокового самоортогонального кода

В физическом канале на передаваемый сигнал действует некоторый шум. В данной работе рассматривается случай наличия только аддитивного «белого» гауссовского шума (канал с АБГШ).

Демодулятор на основе принятого из канала сигнала оценивает значение принятых бит и в некоторых случаях их надежности.

Декодер канала осуществляет декодирование принятых данных, в процессе которого с использованием внесенной кодером избыточности исправляются канальные ошибки. В качестве декодера в данной работе используется МПД, пример схемы реализации которого для СОК, заданного полиномом g=(0, 1, 4, 6), представлен на рис 3.

Приемник данных получает декодированные данные и сравнивает их с переданными для оценки частоты битовых и/или блоковых ошибок после декодера.

Для данной конфигурации системы передачи данных была реализована модель с использованием вычислительных ресурсов центрального процессора (CPU). При этом удалось получить скорость передачи данных в

системе только на уровне 320 кбит/с. Для проведения эксперимента использовался процессор Intel(R) Core(TM) i3 2.53 GHz (двухъядерный процессор). Во время моделирования удалось достичь практически 100 % загрузки обоих ядер CPU. При такой скорости моделирование передачи 10^{12} бит через канал связи займет около 868 часов. Очевидно, что подобные временные затраты являются неприемлемыми.

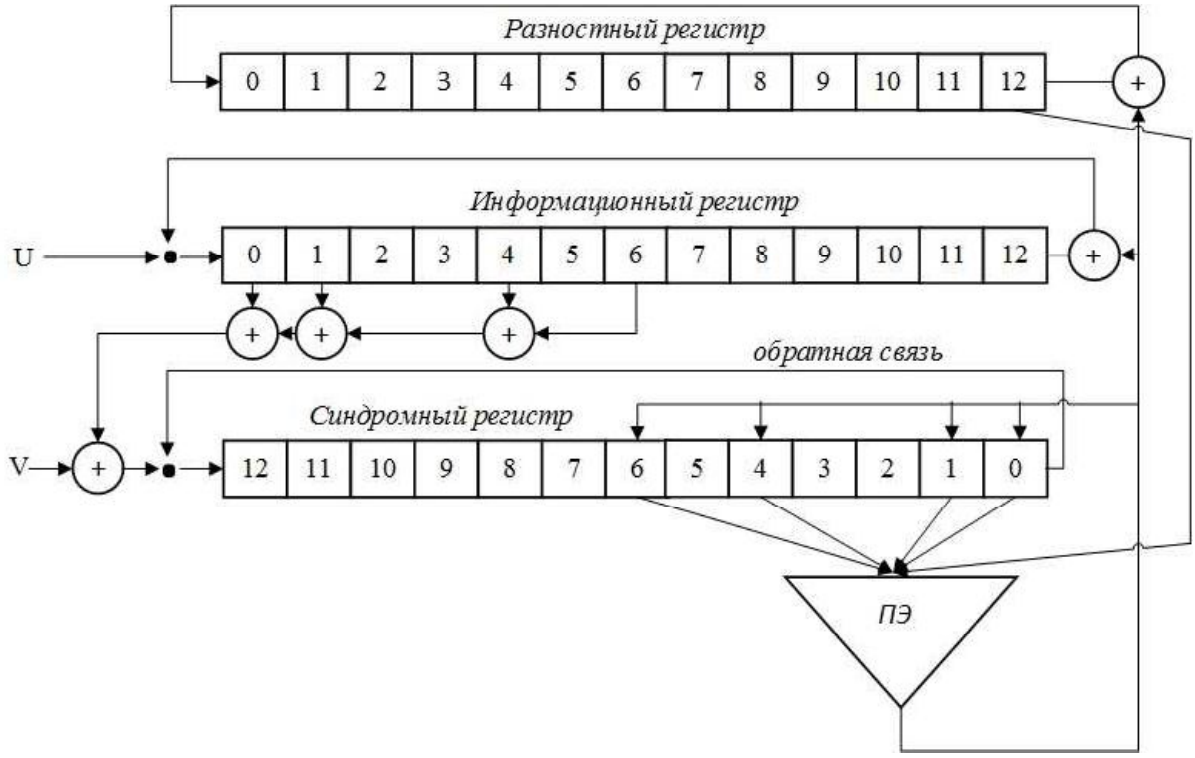

Рис. 3. Схема многопорогового декодера блокового кода

Исходя из вышесказанного, необходимо найти возможность по увеличению скорости моделирования системы передачи данных с МПД. Такая возможность заключается в использовании для моделирования ресурсов GPU, что подробно обосновано в [3].

Использование GPU. В настоящее время для вычислений на GPU активно продвигаются две технологии:OpenCL и CUDA.

OpenCL (Open Computing Language) – это технология, реализующая параллельные компьютерные вычисления на различных типах графических и центральных процессоров [4].

CUDA (Compute Unified Device Architecture) – программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia [5].

Обе технологии представляют схожий функционал для активного и эффективного использования вычислительных ресурсов GPU. Существенным отличием является то, что CUDA поддерживается и продвигается

преимущественно компанией Nvidia, следовательно, для моделирования с использованием CUDA понадобится персональный компьютер с GPU производства именно этой компании [6]. Данный факт существенно сужает парк компьютеров, который может использоваться для моделирования. В то же время OpenCL поддерживается практически всеми производителями GPU (включая Nvidia), ничуть не уступая в эффективности. Исходя из этого выбор OpenCL для моделирования работы является очевидным.

Особенности работы OpenCL. OpenCL имеет ряд архитектурных особенностей, которые специфически отражаются на реализации модели системы передачи данных с МПД. Любая программа на OpenCL содержит в себе две части: хост (host), который запускает расчеты, и OpenCL-часть, которая непосредственно и выполняет вычисления. Архитектура приложения представлена на рис. 4.

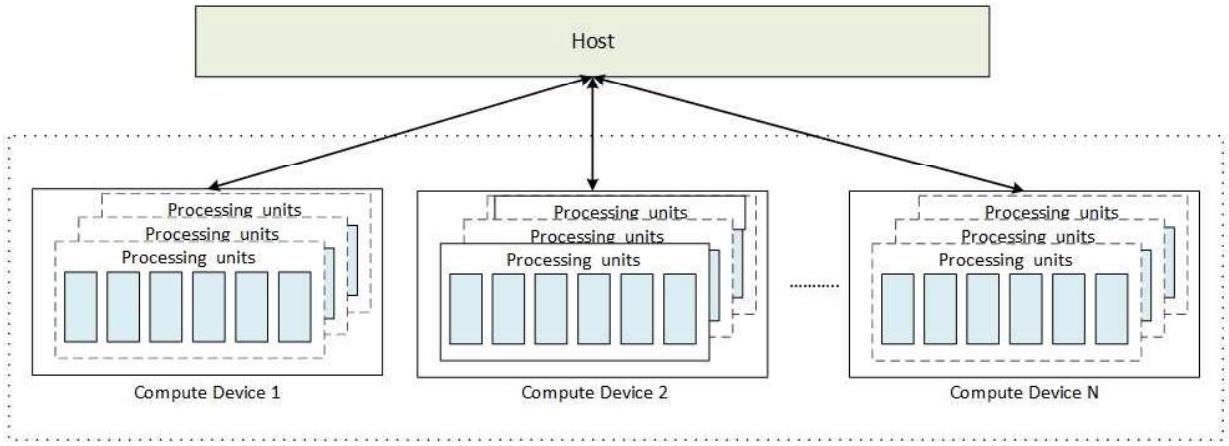

Рис. 4. Архитектура приложения OpenCL

В роли хоста в данной архитектуре будет выступать CPU, а в роли Compute Device (вычислительного устройства) – GPU. Базовая реализация модели будет использовать только одну GPU, но в перспективе можно увеличить их количество.

Распараллеливание вычислений на GPU будет осуществляться за счет вычислительных конвейеров (на рис. 4 они отображены как «Processing Unit»). К примеру, используемая в эксперименте GPU *AMD Radeon HD 6300M Series* содержит 128 конвейеров. Отметим, что практика параллельных вычислений в настоящее время различает два подхода к организации параллельных вычислений (рис. 5) [7]:

параллельная обработка данных (Data Parallel);

параллельное выполнение задач (Task Parallel).

Для реализации работы модели системы передачи данных с МПД необходимо последовательное выполнение значительного объема операций, имеют у них строгую последовательность, нарушение которой приведет к некорректным результатам. Использование подхода Task Parallel в

данном случае не оптимально, так как потребует разложения необходимых операций по сложной схеме, при которой каждая операция будет выполняться в отдельном потоке для всего объёма экспериментальных данных, что приведет к необходимости сдвига выполнения всех операций.

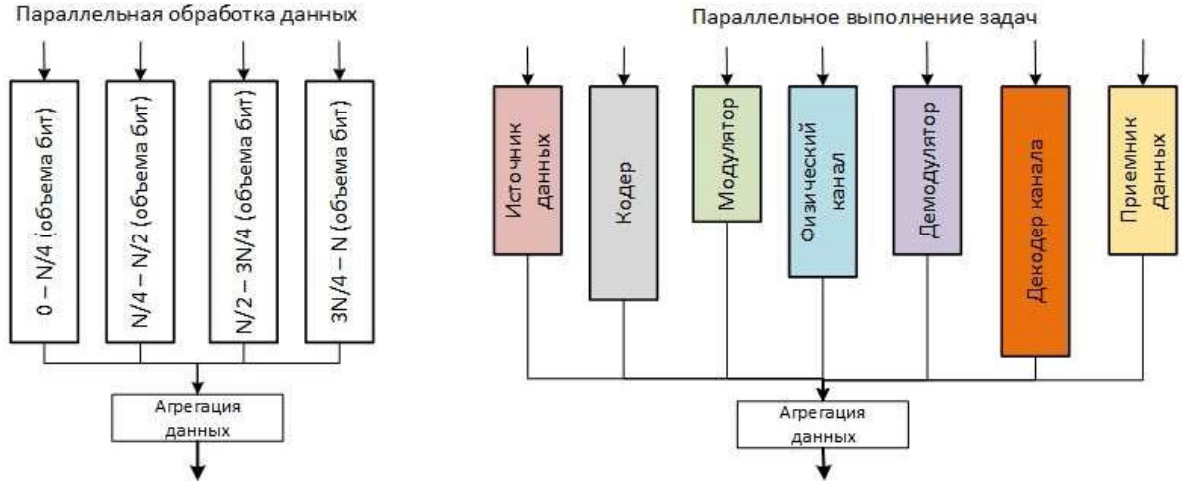

Рис. 5. Подходы к организации параллельной обработки данных

Данный факт станет причиной не максимально эффективного использования вычислительных ресурсов GPU. Кроме того, очевидно, что такая схема усложнит реализацию модели, так как необходимо будет наладить синхронизацию независимых потоков с разными операциями, и, как следствие, конечный результат будет доступен только после завершения работы всех потоков.

В то же время подход Data Parallel позволит применить полный объем необходимых операций для разных наборов данных в отдельном потоке, который будет запущен параллельно, что позволит сконцентрировать весь набор необходимых операций в одном потоке. Это приведет к появлению возможности одновременного запуска всех потоков в момент старта вычислений, увеличив тем самым эффективность использования GPU, что будет достигнуто за счет ликвидации простоя потоков. Кроме того, Data Parallel – подход значительно снизит алгоритмическую сложность реализации модели, так как не потребует сложной синхронизации.

С использованием техники Data Parallel встала задачи разработки модели работы системы передачи данных с МПД на языке OpenCL. Язык программирования OpenCL (также называется OpenCL C) встроен в технологию OpenCL и позволяет программировать ход вычислений на GPU. OpenCLC основан на нотации C99 с рядом существенных ограничений, которые кратко перечислены в [3].

Модель на OpenCL. Реализация модели на OpenCL позволила достигнуть скорости декодирования около 2 Мбит/с при использовании GPU AMD Radeon HD 6300M Series, что является значительным прогрессом по

сравнению с реализацией для CPU и позволяет сократить время моделирования передачи 10^{12} бит до 134 часов, что, однако, является недостаточным результатом. Из-за не вполне удовлетворительных результатов было проведено исследование по обнаружению «бутылочного горлышка», которое ограничивало рост скорости моделирования.

Поиск возможности ускорения модели на OpenCL. Для оптимизации работы и ускорения работы модели необходимо провести исследование затрат, которые приходятся на каждый узел схемы передачи данных. Вычислительные затраты состоят из двух типов операций: элементарных арифметических операций и операций обращения к памяти. Стоит отметить, что обращения к памяти относились только к переменным, размещенным в глобальной памяти.

Источник данных. Так как язык программирования OpenCL не имеет встроенного генератора случайных чисел, то был реализован пользовательский генератор случайных чисел, который требует приблизительно 11 элементарных арифметических операций на генерирование одного бита. При генерировании данных необходимы 2 обращения к памяти.

Кодер канала. Количество элементарных арифметических операций, которое требуются на кодирование сообщения, зависит от образующего полинома, количества информационных ветвей и длины порождающего полинома. К примеру, для кодера с n_k информационными и с n_r проверочными ветвями среднее количество элементарных операций, затраченных на кодирование одного информационного бита, может быть рассчитано по формуле

$$V_C = \frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k}(J_{ij}-1)}{n_k}, \qquad (1)$$

где J_{ij} – число отводов от i-й информационной ветви к j-й проверочной.

Отметим, что в случае, когда число отводов от каждой информационной ветви к сумматору, который вычисляет значение проверочного бита, одинаково и равняется C, формула (1) может быть преобразована к виду

$$V_C = Cn_r - 1 = d - 2,$$

где d – кодовое расстояние, равное Cn_r+1.

Количество операций обращения к памяти для кодирования всего сообщения может быть оценено по формуле

$$U_C = \frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k}2J_{ij}}{n_k} + 1. \qquad (2)$$

Такое число операций обращения к памяти связано с дополнительной необходимостью определения индексов элементов порождающего полинома, который используется при формировании проверочного бита.

В случае, если $J_{ij}=C$, формула (2) преобразуется к виду

$$U_c = 2Cn_r + 1 = 2d - 1.$$

Модулятор. При использовании модуляции типа BPSK на один бит требуются 1 элементарная арифметическая операция и 2 операции обращения к памяти.

Физический канал искажает передаваемый сигнал с помощью генератора случайных чисел. На один бит требуется 13 элементарных арифметических операций и 2 операции обращения к памяти.

Демодулятор. При формировании жесткого решения демодулятора на один бит требуются 1 элементарная арифметическая операция и 2 операции обращения к памяти.

Декодер является самым затратным с вычислительной точки зрения узлом системы передачи данных. При многопороговом декодировании последовательно выполняются следующие шаги.

1. Заполнение синдромного регистра. По вычислительным затратам этот процесс аналогичен процессу кодирования с добавлением одной элементарной арифметической операции:

$$V_s = \frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k} J_{ij}}{n_k}. \qquad (3)$$

В случае, если $J_{ij}=C$, формула (3) преобразуется к виду

$$V_s = Cn_r = d - 1.$$

Число операций обращения к памяти также увеличивается на единицу по сравнению с кодированием:

$$U_s = \frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k} 2J_{ij}}{n_k} + 2. \qquad (4)$$

В случае, если $J_{ij}=C$, формула (4) преобразуется к виду

$$U_s = 2Cn_r + 2 = 2d.$$

2. Вычисление суммы на пороговом элементе (ПЭ) и сравнение суммы с порогом. Количество элементарных арифметических операций на данном шаге вычисляется по формуле

$$V_t = I\left(\frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k} J_{ij}}{n_k} + 1\right), \qquad (5)$$

где I – число итераций декодирования.

В случае, если $J_{ij}=C$, формула (5) преобразуется к виду $V_t = I(Cn_r + 1) = Id$.

Количество операций обращения к памяти может быть вычислено по формуле

$$U_t = I(\frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k} 2J_{ij}}{n_k} + 2). \quad (6)$$

В случае, если $J_{ij}=C$, формула (6) преобразуется к виду

$$U_t = I(2Cn_r + 2) = 2Id.$$

3. Коррекция. Затраты на коррекцию на каждой из итераций декодирования совпадают с затратами на кодирование. В результате число арифметических операций определяется как

$$V_k = Ip(\frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k} J_{ij}}{n_k} + 2), \quad (7)$$

где p – средняя вероятность коррекции бита на текущей итерации в процессе работы МПД.

В случае, если $J_{ij}=C$, формула (7) преобразуется к виду

$$V_k = Ip(Cn_r + 2) = Ip(d + 1).$$

Количество обращений к памяти может быть вычислено как

$$U_k = Ip(\frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k} 2J_{ij}}{n_k} + 2). \quad (8)$$

В случае, если $J_{ij}=C$, формула (8) преобразуется к виду

$$U_k = Ip(2Cn_r + 2) = 2Ipd.$$

Еще раз отметим, что этап коррекции в отличие от первых двух выполняется не для каждого декодируемого бита, а только в случае необходимости его коррекции. Очевидно, что это происходит с вероятностью p, не большей вероятности ошибки в канале. Учитывая область отношений сигнал/шум, в которой МПД может эффективно исправлять ошибки, можно предположить, что коррекция осуществляется не более, чем на 10 % битов. Это позволяет утверждать, что для используемых на практике параметров кода число арифметических операций и число обращений к памяти в расчете на один информационный бит можно сверху оценить как $2I$ (здесь I – число итераций декодирования). При этом данная оценка является существенно завышенной, поскольку на последних итерациях декодирования обычно осуществляется много меньше исправлений битов, чем на первых.

Приемник данных потребует выполнение 1 элементарной арифметической операции и 3 операций обращения к памяти.

Исходя из вышесказанного, общее число элементарных арифметических операций на передачу одного информационного бита будет

$$V_{total} = \frac{\sum_{j=1}^{n_r}\sum_{i=1}^{n_k}((2+I(1+p))J_{ij}-1)}{n_k} + I(2p+1)+28. \quad (9)$$

В случае, если $J_{ij}=C$ формула (9) преобразуется к виду

$$V_{total} = d(2+I(1+p))+Ip+25.$$

Также можно определить суммарное число обращений к памяти

$$U_{total} = \frac{\sum_{j}^{n_r}\sum_{i}^{n_k}4J_{ij}+2IJ_{ij}(1+p)}{n_k} + 14 + I + 2Ip. \quad (10)$$

В случае, если $J_{ij}=C$, формула (10) преобразуется к виду

$$U_{total} = 4d + I(2pd+2d-1)+10.$$

Таким образом, для обсуждаемой системы передачи данных, в которой используется код с кодовым расстоянием $d = 9$ и МПД с 5 итерациями декодирования, количество элементарных арифметических операций на один кодируемый бит будет составлять 93, а количество операций по обращению к памяти будет равно 140.

В табл. 1 представлена доля времени, затраченного на работу отдельных блоков системы передачи данных, полученная в результате компьютерного моделирования. Результаты аналитического расчета числа арифметических операций и числа обращений к памяти для каждого из этих же блоков также представлены в табл. 1. Из таблицы видно, что большую часть времени моделирования занимает декодер канала, хотя по количеству элементарных арифметических операций его «лидерство» не так кардинально. Однако декодер требует наибольшего числа обращений к памяти, что позволяет сделать вывод, что операции обращения к глобальной памяти имеют больший вес, чем арифметические при использовании GPU. Значительные временные затраты на операции по работе с глобальной памяти обусловлены специфической архитектурой организации памяти в OpenCL. Данные, над которыми происходят операции, могут храниться в локальной или в глобальной памяти. Из-за архитектурных особенностей доступ к глобальной памяти занимает значительно больше времени чем к локальной. Локальная память же работает гораздо быстрее. Главной проблемой при работе с локальной памятью является ограничение, связанное с её объемом, который зависит от конкретной модели GPU. Для *AMD Radeon HD 6300M Series* объем доступной локальной памяти составляет 32

Кбайт. Таким образом, число обращений к глобальной памяти для увеличения скорости работы модели следует минимизировать за счет правильного размещения переменных в локальной памяти.

Оптимизация модели на OpenCL. В процессе оптимизации часто используемые массивы были перенесены в локальную память: массив «индексы» (узел кодера), массив «индексы». (узел декодер), массив «пороговые значения» (узел декодера). Это позволило значительно улучшить результаты моделирования и достичь скорости моделирования 11 Мбит/с. Таким образом, по сравнению с реализацией только с использованием CPU скорость моделирования была увеличена в 34 раза. При такой скорости моделирование передачи 10^{12} бит через канал связи с кодированием и декодированием займет всего около 25 часов. Стоит отметить, что скорость моделирования практически не зависит от длины кода (табл. 2).

Таблица 1

Затраты на моделирование одного символа

Узел	% затраченного времени моделирования	Доля элементарных арифметических операций на один бит, %	Доля операций по обращению к памяти на один бит, %
Источник	0.16	11	1
Кодер канала	5.10	7	17
Модулятор	0.20	1	2
Физический канал	0.63	13	2
Демодулятор	0.30	2	2
Декодер канала	**93.25**	**58**	**113**
Приемник данных	0.32	1	3

Таблица 2

Скорость моделирования кодов с различным значением длины кодового слова

Длина кода	Скорость моделирования на CPU, Кбит/с	Скорость моделирования на GPU, Кбит/с
170	258	11250
1700	260	11284
17000	261	11281

Также отметим, что скорость моделирования уменьшается с ростом кодового расстояния используемого кода, но при использовании GPU это уменьшение оказывается несколько меньше, чем при использовании CPU (табл. 3). А поскольку применяемые, в реальных системах передачи и хра-

нения, данных коды могут иметь длину в несколько десятков тысяч битов и кодовое расстояние до нескольких десятков, то преимущество использования GPU над CPU по скорости работы модели при таких параметрах будет еще больше, что видно из табл. 3.

Таблица 3

Скорость моделирования кодов с различным значением кодового расстояния

Кодовое расстояние	Скорость моделирования на CPU, кбит/с	Скорость моделирования на GPU, кбит/с	Прирост производительности GPUпо отношению к CPU
5	337	10214	x30
7	290	9959	x34
9	230	9264	x40
11	207	8520	x40
13	176	7356	x42

Заключение

Полученные результаты дают основание считать перспективным использование предложенного подхода к решению задачи уменьшения временных затрат на ресурсоемкое оценивание характеристик МПД с помощью компьютерного моделирования. Существующие результаты имеют широкий потенциал для дальнейшего алгоритмического улучшения. Так же очевиден легко извлекаемый аппаратный потенциал, который позволит повысить скорость моделирования за счет использования более мощных GPU, которые в настоящее время широко доступны.

Работа выполнена при поддержке РФФИ и гранта Президента РФ.

Список литературы

1. Золотарев В.В., Овечкин Г.В. Помехоустойчивое кодирование. Методы и алгоритмы: справочник. М.: Горячая линия–Телеком, 2004, 126 с.

2. Золотарев В.В., Зубарев Ю.Б., Овечкин Г.В. Многопороговые декодеры и оптимизационная теория кодирования. М.: Горячая линия – Телеком, 2012, 239 с.

3. Демидов Д.С., Овечкин Г.В. Моделирование системы передачи данных с многопороговым декодером с использованием OpenCL // Фундаментальные исследования. 2015. №12. С.243 – 247.

4. The OpenCL Specification. Version: 2.0. Document Revision: 22 / Khronos OpenCL Working Group, Editor: AaftabMunshi, 2014, 483 p.

5. CUDA Programming Guide / 1.2 CUDA: A New Architecture for Computing on the GPU [Электронный ресурс] URL: http://docs.nvidia.com/cuda/index.html#axzz3tRZ1ADO7 (Дата обращения 01.12.2015).

6. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: учебное пособие / А.В. Боресков, А.А. Харламов, Д.А. Марковский, Д.Н. Микушин, Е.В. Мортиков, А.А. Мыльцев, Н.А. Сахарных, В.А. Фролов М: Изд-во Московского университета, 2012. 336 с.

7. Подвальный С.Л., Холопкина Л.В., Попов Д.В. Численные методы и вычислительный эксперимент. Уфа: Уфимский государственный авиационный технический университет, 2005. 224 с.

Демидов Дмитрий Сергеевич, асп., dmitri-demidiv@yandex.ru, Россия, Рязань, "Рязанский государственный радиотехнический университет,

Овечкин Геннадий Владимирович, д-р техн. наук, проф., g_ovechkin@mail.ru, Россия, Рязань, Рязанский государственный радиотехнический университет

DEVELOPMENT OF MODEL FOR COMMUNICATION SYSTEM WITH MULTITHRESHOLD DECODER FOR SELF-ORTHOGONAL CODES WITH OLENCL TECHNOLOGIES

D.S. Demidov, G.V. Ovechkin

The analysis of simulation of communication systems with multithreshold decoders (MTD) for self-orthogonal codes was performed. It's shown to speedup computer model the GPU can be used. The computer model of communication system with MTD based on GPU was developed and possibilities of speedup of the model are reviewed. It's shown the using of suggested model provide a 40x speedup versus equivalent CPU-based implementation .

Key words: communication systems, error correction coding, multithreshold decoding, computer simulation, OpenCL, GPU, CPU.

Demidov Dmitry Sergeevich, postgraduate, dmitri-demidiv@yandex.ru, Russia, Ryazan, Ryazan State Radio-Engineering University,

Ovechkin Gennady Vladimirovich, doctor of technical sciences, professor, g_ovechkin@mail.ru, Russia, Ryazan, Ryazan State Radio-Engineering University