УДК 004.052.2, 004.052.42, 004.052.44, 004.087.2.

ВЫСОКОНАДЁЖНЫЕ ЭЛЕКТРОННЫЕ ИНФОРМАЦИОННЫЕ ХРАНИЛИЩА: ПРОБЛЕМЫ, ЗАДАЧИ, ТЕХНОЛОГИИ

Мороз Владимир Владимирович[1], Добрынин Владимир Николаевич[2], Баулина Людмила Викторовна[3]

[1]Старший преподаватель;
ГБОУ ВПО «Международный Университет природы общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская область, г. Дубна, ул. Университетска, 19;
e-mail: moroz@uni-dubna.ru.

[2]Кандидат технических наук, профессор Института системного анализа и управления;
ГБОУ ВПО Международный Университет природы, общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: arbatsolo@yandex.ru.

[3]Студентка;
ГБОУ ВПО «Международный Университет природы общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская область, г. Дубна, ул. Университетска, 19;
e-mail: L-univ@mail.ru.

С развитием электронных технологий хранения информации возникла проблема управления уровнем надежности этих технологий. В статье проведен обзор состояния по проблеме исследования, поставлена задача решения этой проблемы и предлагается универсальный метод – обеспечения надежности хранения на основе помехоустойчивого кодирования. В качестве помехоустойчивого кода приводится теоретическое описание линейных кодов.

Ключевые слова: устройства/технологии хранения данных, системы обеспечения надежности хранения информации, помехоустойчивое кодирование.

HIGH-RELIABLE ELECTRONIC INFORMATION STORAGES: PROBLEMS, TASKS, TECHNOLOGIES

Moroz Vladimir[1], Dobrinin Vladimir[2], Baulina Ludmila[3]

[1]Senior teacher;
Dubna International University of Nature, Society, and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: moroz@uni-dubna.ru.

[2]Candidate of Science in Engineering, professor of Institute of system analysis and management;
Dubna International University of Nature, Society, and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: arbatsolo@yandex.ru.

[3]Student;
Dubna International University of Nature, Society and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: L-univ@mail.ru.

With development of electronic information storage technologies a reliability layer management problem of these technologies are appeared. In the article the review of a status on a research problem is

carried out, the solution task of this problem is set and the universal method of storage support reliability based on noise proof coding is offered. The theoretical description of the line codes is provided as a noise proof code.

Keywords: devices/data storage technologies, support systems for secure information storage, noiseproof coding.

Введение

Информационное общество характеризуется массовыми технологиями потребления информации. Доступность, качество, надёжность информации, её стоимость – всё это отражает уровень информатизации общества. Огромное разнообразие на рынке аппаратно-программного обеспечения и развитой инфраструктуры сбора, хранения, обработки информации, поддержки и развития информационных систем, порождают и актуализируют известные проблемы, в первую очередь связанные с высокой надёжностью хранения и передачи информации в условиях интенсивно изменяющихся возможностях, поступающих на рынок новых устройств (в том числе и мобильных). Централизованные и мобильные электронные хранилища информации используются в различных научно-прикладных сферах деятельности. Они являются источниками постановок проблем, формулирования задач, поиска пропущенных, углублённых, расширенных и новых знаний. Проблема надёжности хранения информации связана с интервалом времени, в течении которого необходимо обеспечить требуемые качества информации: минимум ошибок, возникающих в процессе хранения; восстанавливаемость, связанная с возможностью исправления ошибок; преемственность, связанная с изменяющимися технологиями производства новых устройств хранения и технологиями приёма/передачи информации. Постоянное повышение производительности вычислительных мощностей за счёт широкого применения параллельных, конвейерных, частичных и квантовых вычислений, а также GRID-технологий акцентирует внимание исследователей на использование этих возможностей при разработке высоконадёжных электронных систем хранения и обработки информации. Приведём пример. На протяжении жизненного цикла социотехническая система проходит точки бифуркации, которые характеризуются возможностью системы переходом в новое качественное состояние или распадом. Переход в новое качественное состояние означает, что в процессе самоорганизации были решены новые возникшие проблемы. Решение новых проблем, как правило, связаны с приобретением новых знаний (или пропущенных, уточнённых, углублённых, расширенных). Если в системе, в процессе её развития, предусмотрены механизмы накопления знаний – эти накопленные знания могут стать источником решения проблемы. Этот пример отражает все проблемы надёжности хранения и эффективной обработки информации. В статье выполнен обзор текущего состояния систем хранения информации и технологий, обеспечивающих требуемую надёжность.

Развитие устройств хранения данных

Информационное общество – общество, в котором большинство работающих занято производством, хранением, переработкой и реализацией информации (знаний). Для обозначения всей информации введено понятие «Цифровая Вселенная» – это весь объем созданной, записанной, реплицированной человечеством цифровой информации. Это цифровые фотографии, электронные тексты, телефонные звонки, аудио- и видео-записи и т.д. Американская корпорация EMC, одна из крупнейших в мире корпораций на рынке продуктов, услуг и решений для хранения и управления информацией, объявила результаты проведенного по ее заказу в 2011 году аналитическим агентством IDC исследования «Цифровой Вселенной» – Extracting Value from Chaos («Как получить пользу от хаоса») [1]. Это проводимое уже четвертый год подряд исследование, в ходе которого оценивается и прогнозируется объем ежегодно создаваемой и копируемой цифровой информации, а также анализируется ее влияние на потребителей, корпоративный сектор и ИТ-специалистов. Опубликованные результаты свидетельствуют об огромных объемах создаваемой, копируемой цифровой информации, так же о стремительных темпах их роста. Согласно отчету, объем информации во всем мире увеличивается более чем в два раза каждые два года – быстрее, чем по закону Мура. Источников большого количества данных в современном информационном обществе великое множество. В их качестве могут выступать непрерывно поступающие данные с

измерительных устройств, события от радиочастотных идентификаторов, потоки сообщений из социальных сетей, метеорологические данные, данные дистанционного зондирования земли, потоки данных о местонахождении абонентов сетей сотовой связи, устройств аудио- и видеорегистрации. Собственно, массовое распространение перечисленных выше технологий и принципиально новых моделей использования различно рода устройств и интернет-сервисов послужило отправной точкой для проникновения большого количества данных едва ли не во все сферы деятельности человека. В первую очередь, научно-исследовательскую деятельность, коммерческий сектор и государственное управление.

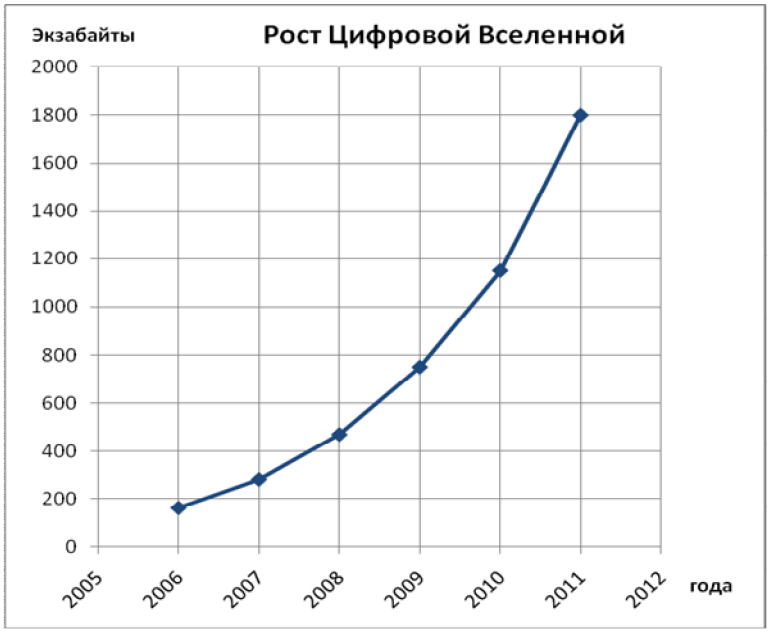


Рис. 1. Рост цифровой вселенной (1 экзабайт = 1 млрд. гигабайт =10^{18} байт)

Основные стимулы такого непрерывного роста – прогресс технологий и деньги. Новые технологии «упрощения информации» снизили стоимость создания, сбора, классификации информации и управления ею. Кроме того, ежегодные инвестиции корпоративного сектора в «Цифровую Вселенную» с 2005 года (включая облака, оборудование, программное обеспечение, сервисы и персонал для создания информации, управления ею и извлечения с ее помощью прибыли) выросли на 50%. Исследователи пришли к выводу, что численность ИТ-персонала не соответствует быстрому росту числа серверов, потребностей в управлении данными и объемов файлов. В следующие десять лет ИТ-департаменты по всему миру могут столкнуться со множеством проблем: серверов станет в 10 раз больше, объем информации, которой нужно управлять вырастет в 50 раз, в 75 раз вырастет число файлов или контейнеров, в которых информация инкапсулируется в «Цифровой Вселенной», а численность ИТ-персонала, который управляет всей этой информацией и серверами, вырастет всего в 1,5 раза. Число этих объектов будет расти даже быстрее, чем объем хранимой в них информации, поскольку будет использоваться все больше встроенных систем, например, интеллектуальные датчики в одежде, мостах, медицинских устройствах. «Хаотичный рост объемов информации открывает новые бесчисленные возможности, связанные с фундаментальными изменениями в обществе, технологиях, науке и экономике. Большие объемы данных меняют подход бизнеса к управлению информацией, которая является самым важным ресурсом предприятия, и извлечению из нее ценности» – Джереми Бертон, директор по маркетингу корпорации EMC.

IDC предсказывает, что к 2020 году объемы цифровой информации вырастут в 30 раз, в то время как число профессионалов в сфере ИТ, способных работать с ней, возрастет лишь в 1,4 раза. Также, согласно выводам IDC, организация, насчитывающая 1000 информационных работников, ежегодно теряет 6 миллионов долларов США из-за того, что сотрудники занимаются поиском, восстановлением и переформатированием информации. Так же, один из важных выводов исследования: «Менее трети информации в «Цифровой Вселенной» имеет какую-либо защиту и только половина важной информации защищена». Современный мир характеризуется такой

интересной тенденцией, как постоянное повышение роли/ценности информации. Известно, что производственные процессы состоят из материальной и нематериальной составляющих. Материальная – это необходимые для производства оборудование, материалы, некоторая энергия. Нематериальная составляющая – технология производства. Сегодня, с приходом информационного общества, появилось множество отраслей производства, которые почти на 100% состоят из одной информации, например, дизайн, создание программного обеспечения, реклама и многое другое. Так же, развитие информационных технологий затронуло практически все сферы человеческой деятельности [2].

Анализ современных технологий хранения данных: быстродействие, объем, надёжность

Взаимодействие с информацией в информационном обществе немыслимо без носителей информации. Информация не может существовать сама по себе, в отрыве от материального носителя, которые, как известно, постоянно совершенствуются. Носитель информации – устройство, предназначенное для записи, хранения и считывания информации, однако в настоящий момент данный термин не описывает все возможные функции современных носителей. История носителей информации началась достаточно давно. Еще в 1725 году Базиль Бушон (французский изобретатель) создал текстильный станок с перфолентой, которая позволяла воспроизводить заданный рисунок на ткани.

С развитием общества, развитием технологий появлялись новые средства записи и хранения данных. В современном мире очень высока ценность информации. Сегодня существуют информационные отрасли, например, реклама, создание программного обеспечения, дизайн, и т.д. Соответственно значение и ценность/стоимость информации постоянно увеличивается.

Понятно, что информация может существовать только на носителях, которые постоянно изменяются и совершенствуются.

Рассмотрим доступные сегодня технологии хранения данных.

Оптические диски

Оптический диск – носители информации, выполненные в виде дисков, чтение с которых ведётся с помощью оптического излучения. Основа диска сделана из поликарбоната, на который нанесён специальный слой, который служит для хранения информации. Для считывания информации используется обычно луч лазера, который направляется на специальный слой и отражается от него. При отражении луч модулируется мельчайшими выемками, так называемыми «питами» (от английского *pit* — «ямка», «углубление») на специальном слое, на основании декодирования этих изменений устройством чтения восстанавливается записанная на диск информация.

Существует немало оптических дисков, и вообще их можно разделить на несколько поколений.

1-ое поколение:

Лазерный диск впервые был выпущен в 1978 году. Он содержит аналоговое видео в композитном представлении и звуковое сопровождение в аналоговой и/или в цифровой форме. На таком диске может содержаться до 5 минут видео и до 20 минут аудио.

Компакт диск (Compact Disc – CD) был разработан 1979 году компанией Sony. Емкость составляет от 650 Mb до 900 Mb.

Магнитооптический диск – носитель данных, сочетающий в себе свойства магнитных и оптических носителей данных. Впервые появился в 1980 году. Первые магнитооптические диски были размером с 5,2 дюйма, потом появились диски размером 3,5 дюйма. Запись на такой диск осуществлялась следующим образом: лазер разогревает участок дорожки выше точки Кюри, после этого электромагнитный импульс изменяет намагниченность, создавая отпечатки, похожие на «пииты» на оптических дисках. Считывание осуществляется тем же самым лазером, но на меньшей мощности, недостаточной для разогрева диска.

Такой диск слабо подвержен механическим повреждениям, магнитным полям, количество циклов перезаписи около 10 млн. Однако, такие недостатки, как низкая скорость записи, высокое энергопотребление, высокая цена дисков и приводов, сделали такие диски малораспространенными и невозможными для повсеместного применения, поскольку нет ни одного стандарта на эти накопители и приводы для них.

2-ое поколение:

DVD – диск аналогичный CD, но имеющий более плотную структуру рабочей поверхности, что позволяет хранить и считывать больший объём информации. На сегодняшний день можно найти DVD диски емкостью от 1,46 Гб до 17,08 Гб.

MiniDisc Был разработан и впервые представлен компанией Sony в 1992 году. Позиционировался как замена компакт-касетам, к тому времени уже полностью изжившим себя.

Fluorescent Multilayer Disc – разработан компанией «Constellation 3D», использующий флуоресценцию вместо отражения для хранения данных. Форматы, основанные на измерении интенсивности отраженного света (такие как CD или DVD), имеют практическое ограничение в 2 слоя хранения данных, главным образом, из-за эффекта интерференции. Однако использование флуоресценции позволяет работать, соответствуя принципам объёмной оптической памяти и иметь до 100 слоёв. Они позволяют вместить объём до 1 Тб при размерах обычного компакт-диска.

GD-ROM – разработано компанией Yamaha. Он подобен стандарту CD-ROM за исключением того, что биты на диске упакованы плотнее, обеспечивая более высокую емкость (приблизительно 1.2 Гб, что почти вдвое больше емкости типичного CD-ROM).

Universal Media Disc – разработан компанией Sony для использования в игровых приставках PlyStation Portable. Может вмещать до 1,8 Гб данных, в частности: игры, видео и музыку.

3-е поколение:

Blu-ray Disc впервые выпущен в 2006 году. Однослойный диск Blu-ray (BD) может хранить 25 Гб, двухслойный диск может вместить 46,6 Гб (50 Гб), трёхслойный диск может вместить 100 Гб, четырёхслойный диск может вместить 128 Гб. Ещё в конце 2008 года японская компания Pioneer демонстрировала 16-ти и 20-слойные диски на 400 и 500 Гб, способные работать с тем же самым 405-нм лазером, что и обычные BD-плееры. Компания Pioneer Electronics уже представила привод поддерживающий трёхслойный диск 100 Гб и четырёхслойный диск 128 Гб.

HD DVD – формат цифровых носителей на оптических дисках, предназначенный для хранения видео высокой чёткости и другого высококачественных мультимедийных данных. На одном слое HD VMD-диска помещается до 5 Гб данных, но за счёт того, что диски являются многослойными (до 20 слоёв) их ёмкость достигает 100 Гб.

Ultra Density Optical (UDO) – специальный формат для хранения видео высокой чёткости. UDO представляет собой картридж в 5.25 дюймов с оптическим диском внутри. Объём диска на данный момент составляет от 60 Гб до 120 Гб, а максимально, объём диска может достигать 500 Гб. В этом обзоре представлены не все существующие оптические диски. Большинство из них используется только в очень узком круге (например, игры, хранение фильмов и т.п.). Данные можно записать на оптический диск, в зависимости от его типа 1 или более раз, и затем использовать их. Для активной работы с данными, их редактирования, постоянного перезаписывания, оптические диски не подходят. Первоначально оптические диски были разработаны для резервного копирования данных, переноса.

Магнитные диски

Гибкий магнитный диск или дискета – диск, покрытый слоем ферромагнетика, помещенный в пластиковый корпус. Емкость дискеты составляет от 80 Кб до 2,88 Мб. К середине 90-х появились дискеты Iomega Zip, которые достигали объемов 750 Мб. Но они не смогли вытеснить стандартные трехдюймовые дискеты емкостью всего лишь в 2,88 Мб из-за высокой цены дискет и приводов, а так же из-за особенностей привода ломаться после вставки нерабочих дискет.

Одна из проблем дискет –недолговечность хранения информации. Магнитный диск может размагнититься от воздействия намагниченных предметов, что делало хранение данных на дискетах

ненадежным. Пластиковый корпус дискеты не служил надежной защитой гибкого диска от механических повреждений. Срок службы носителя зависит не только от условий эксплуатации, а от исходного качества. Изделия высокого качества (крупных компаний) выдерживают при эксплуатации до 70 млн. циклов перезаписи, что означает примерно 20 лет активной эксплуатации носителя.

Дискеты были массово распространены с 1970-х и до конца 1990-х годов, уступив более емким и удобным накопителям.

Стриммер (ленточный накопитель) – запоминающее устройство, работающее по принципу магнитной записи на ленточном носителе с последовательным доступом к данным, по принципу действия аналогичен магнитофону. Основное назначение устройства – архивация и резервное копирование данных.

Магнитная лента впервые была использована для записи компьютерных данных в 1951 году. В ЭВМ до широкого распространения жестких дисков использовались накопители на магнитной ленте, аналогичные стримерам, как основной накопитель информации.

Собственно картриджи к ним, на которых хранятся данные, имеют максимальную емкость в 4 Тб.

Основные характеристики картриджей: срок хранения картриджа от 15 до 30 лет; объем до 3 Тб; скорость чтения/записи 2-3 Мб/с; 5000 стандартных загрузок/выгрузок картриджа; приблизительно 260 полных циклов записи ленты.

Накопитель на магнитной ленте, поддерживающий работу одновременно с несколькими лентами, называется ленточной библиотекой. Роботизированные ленточные библиотеки могут содержать хранилища с тысячами магнитных лент, из которых робот автоматически достаёт требуемые ленты и устанавливает в одно или несколько устройств чтения-записи. С программной точки зрения такая библиотека выглядит, как один накопитель с огромной ёмкостью и значительным временем произвольного доступа. Кассеты в ленточной библиотеке идентифицируются специальными наклейками со штрихкодом, который считывает робот. К 2010 году коммерчески были доступны модели ленточных библиотек с ёмкостью до 70 петабайт при использовании 70 000 кассет [3].

Стриммер предназначен для архивирования и резервного копирования больших объемов данных, так как одним из его недостатков является малая скорость передачи информации.

Жесткие диски

Накопитель на жестких магнитных дисках является основным накопителем данных в большинстве компьютеров. Данные записываются на жесткие алюминиевые или стеклянные пластины, которые покрыты слоем ферромагнитного материала, т.е. на магнитные диски. В жестких дисках используется от одной до нескольких таких пластин на одной оси. Считывающие головки не касаются поверхности пластин во время работы благодаря потоку воздуха, образующегося у поверхности при быстром вращении дисков. Отсутствие механического контакта обеспечивает долгий срок службы такого накопителя. Когда диски не вращаются, головка находится за пределами диска в безопасной зоне, где исключен непредвиденный контакт головки и дисков.

Время произвольного доступа варьируется от 2,5 до 16 мс, т.е. это среднее время, за которое жесткий диск выполняет операцию позиционирования головки чтения/записи на произвольный участок диска.

С момента создания первых жестких дисков возможная емкость постоянно увеличивается. На сентябрь 2011 года емкость жесткого диска достигает 4000 Гб (4 Тб) и близится к 5 Тб. Компания Western Digital объявила о начале поставок 3,5-дюймовых жестких дисков для использования в корпоративных системах: WD RE SAS и WD RE SATA емкостью 4 Тб – с интерфейсами SAS и SATA со скоростью передачи данных 6 Гбит/с. Диски состоят из 5 пластин емкостью по 800 Гб. Емкость десяти 60-дисковых модулей 4U, собранных из новых дисков, способна достигать 2,4 Пб [4].

Надежность жестких дисков

Производители жестких дисков вкладывают высокий запас прочности, однако, каким бы надежным ни был жесткий диск, срок его эксплуатации ограничен. Для современных жестких дисков он составляет от 3 лет, и дальше как повезет)) до 7-10 лет. Надежность и производительность так же зависят от микропрограммы жесткого диска, выполняющегося на микропроцессоре накопителя.

Кроме того, жесткие диски могут выйти из строя из-за механических повреждений магнитной поверхности, к примеру из-за падения. Даже падение с небольшой высоты, такой как 20 см может вывести его из строя (что уж говорить о падении на пол). Работающий диск более чувствительный. Даже резкое смещение во время работы может привести к соприкосновению пластины и головки, а как следствие к возникновению царапин и дефектов. Перегрев во время работы так же способствует быстрому «старению» жесткого диска.

Большинство современных жестких дисков поддерживают технологию S.M.A.R.T. (технология оценки состояния жесткого диска встроенной аппаратурой самодиагностики, а так же механизм предсказания времени выхода его из строя), что позволяет продлить срок службы жесткого диска.

Жесткие диски сегодня очень популярны в качестве переносного носителя информации большого объема.

Твердотельные диски

Сегодня пользуются большой популярностью твердотельные диски – носители данных на основе ячеек флэш-памяти. Они встречаются практически везде: в планшетах, смартфонах, фотоаппаратах, ноутбуках, персональных компьютерах. Твердотельные диски обеспечивают довольно высокую производительность в сочетании с мощной аппаратной частью.

Зачастую любители твердотельных дисков, энтузиасты избавляются от старых винчестеров, отдавая предпочтение твердотельным дискам. Однако изначально, флэш-память разрабатывалась не для хранения больших объемов данных. Для полупроводниковой памяти она предоставляет не высокую скорость работы. Так же нужно помнить об износе флэш-памяти в процессе работы.

Увеличение плотности записи на флэш-память, что приведет к уменьшению размеров памяти, потребует значительных усилий и материальных ресурсов.

Срок службы и энергоэффективность почти не поддаются регулированию. Причина этого в принципе организации и работы флэш-памяти.

В общем, флэш-память представлена транзистором с тремя контактами: источником питания, управляющей цепью и стоком. Управляющая цепь пропускает ток, что соответствует значению «1» или блокирует, что соответствует значению «0». Флэш-память устроена примерно так же как и оперативная память, но разница в том, что оперативная память теряет данные после отключения питания, флэш-память же должна сохранять данные. Из-за этого ячейка флэш-памяти имеет дополнительный элемент – плавающий затвор, который имеет способность хранить заряд в течение продолжительного времени. Заряд сообщается плавающему затвору путем подачи напряжения в 10-20 В. Чтение осуществляется с помощью измерительного тока небольшой силы. Если он проходит от истока к стоку, плавающий затвор теряет заряд, и ячейка принимает значение «1». Если ток измерения блокируется, т.е. плавающий затвор заряжен, это соответствует значению «0».

Одна из проблем накопителя на флэш-памяти – это потребность высокого напряжения для записи данных или для их удаления из ячеек. Сам запоминающий элемент плавающего затвора плохо удерживает заряд, поэтому ему нужен толстый слой изоляции, который можно было бы преодолеть только при высоком напряжении. Таким образом, снижается скорость доступа, ведь требуется время для выработки высокого напряжения. Так же, уменьшается срок службы ячеек памяти, ведь при каждой операции записи или удаления теряется некоторая небольшая часть изолирующего слоя.

В среднем, доступные сегодня на рынке SSD-накопители рассчитаны на 10 тысяч операций перезаписи, после чего, они, вероятно, выходят из строя.

Подобная «чувствительность» ячеек флэш-памяти требует дорогих контроллеров, отточенной технологии взаимодействия с ячейками памяти, что сказывается на скорости и энергопотреблении.

Уменьшение площади занимаемой флэш-памятью приводит к обострению проблемы срока службы, ведь толщина изоляционного слоя сокращается.

В качестве последователей твердотельных накопителей на основе флэш-памяти рассматриваются различные технологии хранения данных, которые сейчас проходят исследования, а некоторые из них начинают использовать на практике.

В разработке и исследованиях новых технологий хранения данных участвуют немногие компании. Среди них такие крупные, как IBM, Toshiba, Fujitsu. Понятно, что небольшое количество компаний, имеют достаточно ресурсов (для изучения различных способов создания модулей памяти, обладающих улучшенными возможностями: более высокой скоростью чтения и записи данных, надежностью хранения, энергоэффективностью) для подобных исследований.

Уже существует несколько технологий, которые имеют шансы выйти на рынок в 2013 году, (достойных огласки). Рассмотрим их подробнее [5].

Накопитель S0N0S

Данная технология разработана компаниями Philips и Spansion. Для записи требуется напряжение в два раза меньшее, чем во флэш-памяти. Таким образом, накопитель на основе SONOS может выдержать в 1-10 тысяч раз больше циклов перезаписи. Ячейки памяти имеют такую же структуру, только запоминающий элемент не из кремния, как во флэш-памяти, а из нитрида кремния. Этот материал обладает лучшими характеристиками по сравнению с кремнием, что позволяет прочнее удерживать заряд, и изоляционный слой может быть тоньше, что обеспечивает компактность и простоту технологии производства.

Для удаления данных требуется низкое напряжение в 5-8В, таким образом, затрачивается в 2 раза меньше времени на его выработку, чем во флэш-памяти.

Однако, для рынка размеры, стоимость производства не достигли оптимальных значений.

Возможно, когда дальнейшее уменьшение размеров флэш-памяти станет невозможным, на рынке появится SONOS.

Накопитель FeRAM

Технология разрабатывается в компаниях Ramtron, Texas Instruments, Fujitsu.

По имеющейся информации, ячейка на основе FeRAM выдерживает около 10 квадриллионов (10^{15}) циклов перезаписи, т.е. она практически вечна. Данный вид памяти сохраняет данные, смещая атомы – операция, которая, по идее, может производиться неограниченное количество раз.

Так же достоинства данного вида памяти в том, что для записи данных не нужно высокое напряжение. В сравнении с флэш-памятью данный вид памяти требует вдвое или в четверть более низкое энергопотребление.

Переключение к ячейке памяти происходит много быстрее, чем во флэш-памяти. 1 бит в памяти FeRAM записывается за 0,15 микросекунд, в то время как во флэш-памяти, эта же операция занимает 10000 микросекунд.

Модули памяти производятся для микроконтроллеров компаниями Fujitsu и Texas Instruments. Сегодня стоимость хранения бита информации в такой памяти чрезвычайно высока, поэтому данная память используется только в некоторых областях, как например, в медицинской технике, в системах управления подушками безопасности.

Накопитель MRAM

Носителем информации в MRAM памяти являются магнитные моменты, которые обеспечивают высокую скорость переключения, способные длительное время сохранять свое состояние и изменять его практически неограниченное количество раз.

MRAM память обеспечивает скорость записи до 1000 раз более высокую, чем флэш-память, благодаря короткому времени отклика.

К примеру, для копирования фильма размером 8 Гб на диск MRAM потребовалось бы 0,02 секунды, вместо 21.

Однако существует серьезный недостаток такой памяти: при рабочей частоте более 400 МГц магниты оказывают взаимное влияние друг на друга, что делает невозможным работу с данным носителем.

Данная проблема была решена путем объединения нескольких ячеек памяти в один запоминающий элемент, благодаря чему производительность возросла в 5 раз.

Производство памяти MRAM уже налажено. Этот вид памяти сегодня находит применение лишь в специальных областях, например в космонавтике.

Есть информация о том, что Toshiba, IBM и NEC возможно запустят производство данной памяти в ближайшие годы.

Вообще запоминающие устройства будущего – компактные, довольно быстрые и дешевые устройства.

Технологии увеличения надежности, быстродействия, объема носителей данных

Управление надежностью в рамках конкретного устройства хранения невозможно.

Для увеличения надежности хранения данных существуют такие решения как RAID, копирование, резервирование данных (backup), Hot Spare (Горячий резерв).

Рассмотрим подробнее основные технологии.

RAID

Эта технология использует дисковый массив – набор дисковых устройств, работающих вместе, для повышения скорости/надежности системы ввода/вывода. Дисковым массивом управляет RAID-контроллер (контроллер массива), который содержит в себе функции размещения данных по массиву, а для остальной системы позволяет представлять дисковый массив как одно логическое устройство ввода/вывода. За счет параллельного выполнения операций чтения и записи на нескольких дисках, массив обеспечивает повышенную скорость обменов по сравнению с одним большим диском.

Так же, массивы могут обеспечивать избыточное хранение данных, для того, чтобы данные не были потеряны при выходе из строя одного из дисков. В зависимости от уровня RAID, проводится или зеркалирование или распределение данных по дискам.

Каждый из четырех основных уровней RAID использует свой уникальный метод записи данных на диски, поэтому все уровни обеспечивают различные преимущества.

Уровень RAID 0:

- повышается пропускная способность последовательного ввода/вывода за счет одновременной загрузки нескольких интерфейсов;
- снижается латентность случайного доступа; несколько запросов к различным небольшим сегментам информации могут выполняться одновременно.

Уровень RAID 0 предназначен исключительно для повышения производительности, и не обеспечивает избыточности данных. Поэтому любые дисковые сбои потребуют восстановления информации с резервных носителей.

Уровень RAID 1:

- высокий уровень надежности;
- обеспечивает избыточность хранения информации;

- низкая латентность при чтении.

Уровень RAID 2 и 3:

- параллельная работа всех дисков;
- хранение битов четности для каждого элемента информации;
- простое восстановление данных из соседних дисков и диска с информацией четности;
- для больших объемов данных производительность велика.

Уровень RAID 4 и 5:

- улучшение уровня RAID 3: информация четности распределяется по всем дискам массива;
- возможны одновременные операции чтения и записи;
- подходит для приложений, работающих с небольшими объемами данных (например, система обработки транзакций).

Таблица 1. Сравнительные характеристики технологии RAID разных уровней

Уровень RAID	Механизм обеспечения надежности	Эффективная емкость*	Производительность	Область применения
0	—	100%	Высокая	Приложения без существенных требований к надежности
1	Зеркалирование	50%	Высокая или средняя	Приложения без существенных требований к стоимости
3	Четность	80%	Средняя	Приложения, работающие с большими объемами данных
5	Четность	80%	Средняя	Приложения работающие с небольшими объемами данных (транзакции)

* — доступная пользователю часть объема массива из общего объема дисков в массиве.

Резервное копирование данных (backup)

Существуют специальное программное обеспечение, позволяющие создавать резервные копии данных для домашних персональных компьютеров, либо в некоторой локальной сети, такие как: BackupFly, Acronis Backup & Recovery 10 Workstation, Paragon Drive Backup 10 Workstation, Handy Backup Server, GFI Backup Business. Каждый программный продукт предоставляет различные функциональные возможности.

У резервного копирования данных есть свои преимущества. Оно позволяет надежно защитить данные от утери, так же позволит быстро перенести данные с одного компьютера на другой.

Виды резервного копирования:

Полное резервирование (Full backup). Затрагивает всю систему и все файлы. В течение полного резервирования, копируются все желаемые файлы.

Дифференциальное резервирование (Differential backup). Каждый файл, который был изменен с момента последнего полного резервирования, копируется каждый раз заново. Дифференциальное резервирование ускоряет процесс восстановления. Все, что необходимо для восстановления, это последняя полная и последняя дифференциальная резервная копия. Популярность данного вида резервирования растет, так как все копии файлов делаются в определенные моменты времени, что важно, например, при заражении вирусами.

Инкрементное резервирование (Incremental backup). При таком типе резервирования происходит копирование только тех файлов, которые были изменены с тех пор, как в последний раз выполнялось полное или добавочное резервное копирование. Последующее добавочное резервирование добавляет только файлы, которые были изменены с момента предыдущего добавочного резервирования. В среднем, добавочное резервирование занимает меньше времени, так как копируется меньшее количество файлов. Однако, процесс восстановления данных занимает больше времени, так как должны быть восстановлены данные последнего полного резервирования, плюс данные всех последующих добавочных резервирований. В отличие от дифференциального резервирования, изменившиеся или новые файлы не замещают старые, а добавляются на носитель независимо.

Резервирование клонированием. Клонирование позволяет скопировать целый раздел или носитель (устройство) со всеми файлами и директориями в другой раздел или на другой носитель. Если раздел является, к примеру, загрузочным, то клонированный раздел тоже будет загрузочным [6].

Резервирование в виде образа. Образ – точная копия всего раздела или носителя (устройства), хранящаяся в одном файле [7].

Резервное копирование в режиме реального времени. Резервное копирование в режиме реального времени позволяет создавать копии файлов, директорий и томов, не прерывая работу, без перезагрузки компьютера.

Hot Spare (Горячий резерв)

Hot Spare – технология резервирования электронного оборудования, в которой резерв подключен к системе и либо автоматически заменяет вышедшую из строя компоненту, либо без остановки в работе системы. Зачастую данная технология используется вместе с RAID-массивами.

Так же выделяют такие технологии как *холодный резерв*, т.е. когда требуется ручное подключение оборудования, а так же *теплый резерв* – компоненты требуют ручной замены, однако не требуют остановки системы.

Системы хранения данных

Постоянно растущие объемы данных, возрастающие требования к надежности хранения, быстродействию доступа к данным делают необходимым выделение средств хранения в отдельную подсистему вычислительного комплекса.

Существует такое понятие СХД – Сеть Хранения Данных или Система Хранения Данных. Это специализированное железо и программное обеспечение для работы с большими объемами важной/ценной информации.

Системы хранения данных состоят, как правило, из большого количества устройств хранения (дисковые массивы, ленточные библиотеки). Однако «система хранения данных – это не только диски» [8]. И здесь возникает проблема: согласно оценкам психологов, человек способен эффективно управлять лишь ограниченным количеством объектов от 5 до 9 [9]. Таким образом, при управлении системой хранения данных не обойтись без вспомогательных средств (помощников). В таком качестве выступают описанные выше технологии увеличения надежности, быстродействия и объема хранилищ. Существует еще ряд средств:

Виртуализация – один из путей упрощения управлением системой хранения данных. В самом деле, количество управляемых объектов исчисляется тысячами, причем система работает круглосуточно. К примеру, для добавления дискового пространства может потребоваться остановить систему на несколько часов для подключения новых дисков, создания файловой системы, внесения необходимых изменений.

Современные средства визуализации позволяют администратору управлять системой в терминах дискового пространства, не переходя на уровень железа.

Однако, как правило, такие возможности жестко привязаны к конкретным системам хранения определенных производителей.

Защита данных

Управлять централизированными системами хранения проще. Это подтверждает существующая сегодня тенденция в современных системах хранения данных – создание единых хранилищ данных. Несмотря на то, что защита самих данных от сбоя оборудования контролируется вышеперечисленными методами, такими как технология RAID, hot-spare, резервное копирование, этого недостаточно для достижения необходимого уровня безопасности. Причинами сбоев могут быть не только сбои оборудования, но и человеческие ошибки, бытовые аварии, т.о. необходимо предпринимать комплекс мер по защите данных на всех уровнях.

Дублированные интерфейсы

Дабы исключить единственную точку сбоя в системе дублируются интерфейсы. В зависимости от используемого программного обеспечения имеется возможность разделять нагрузку между интерфейсами, а так же поддерживать непрерывную доступность данных при сбое одного из контроллеров.

Моментальные копии данных

Существует такая опасность, как сбой приложения, который может привести к потере данных. Это решается с помощью создания моментальных копий данных, копированием целостной информации в определенные особым образом для каждой системы моменты времени.

Удаленные копии данных

Для защиты от серьезных аварий (крупный сбой питания, авария здания, его инфраструктуры) создаются копии данных на удаленных хранилищах. Для этого к хранилищу проложены кабели и данные могут передаваться по нескольким протоколам.

Учет и контроль

Для качественного управления системой хранения необходимо знать, сколько данных хранится, тенденции увеличения объемов, хватит ли текущего запаса свободного места. Зная ответы на эти вопросы можно говорить об управляемой системе хранения, которая сможет обеспечить должный уровень сервиса. Получение подобной информации можно реализовать, так сказать, подручными средствами, либо воспользоваться готовым средствами, например, от Sun Microsystems: Sun StorEdge Resource Manager.

Хранение данных в облаках

Облака так же предоставляют всевозможные другие сервисы не только хранение больших объемов данных, но в рамках данной статьи мы рассмотрим облака только с позиции хранилищ данных.

Вообще облачных хранилищ достаточно много. Каждое предоставляет различные возможности, различный объем хранилища для каждого пользователя.

К примеру, решения компании Google: Docs и Drive, как развитие Docs. Google Docs – бесплатный онлайн-офис, включающий в себя текстовый, табличный процессор и сервис для создания презентаций, а также интернет-сервис облачного хранения файлов с функциями файлообмена, разрабатываемый компанией «Google». Это бесплатная программа, работающая в рамках веб-браузера без установки на компьютер пользователя. Объем для хранения файлов, преобразованных в формат документов Google, не ограничен, для остальных форматов сервис Google Docs предоставляет 1 Гб дискового пространства бесплатно. Дополнительный объем можно приобрести.

Раньше компании, имеющие потребность в хранении больших объемов данных, были вынуждены покупать сервера или даже создавать центр обработки данных. На сегодня существуют решения, позволяющие перейти к использованию облачных сервисов хранения и платить только за то, что реально используется. На этом пути существуют преграды (ограничения), так как нужно обеспечить постоянную доступность данных, их сохранность и обработку. К примеру, один из облачных сервисов хранения данных разрабатывается сейчас: Vision Cloud. Создание сервиса рассчитано приблизительно на 3 года: с октября 2010 по сентябрь 2013 года [10].

«В облачных решениях зачастую используются многоуровневые системы хранения. Гипервизор (универсальное ПО для виртуализации ресурсов хранения данных) систем хранения позволяет интегрировать ресурсы iSCSI, Infiniband, FC (Fibre Channel) и FCoE (Fibre Channel over Ethernet), поддерживает не только жесткие (Hard-Disk Drive, HDD), но и твердотельные (Solid State Disks, SSD) диски, а новейшие поколения способны использовать и ресурсы устройств NAS. В результате упрощается внедрение решений высокой доступности для общих файловых кластеров из хостов CIFS и NFS» [11].

Однако хранение данных так же не обходится без недостатков. Конфиденциальность данных хранимых на «публичных» облаках вызывает много споров. На сегодняшний день нет технологии, которая бы гарантировала 100% конфиденциальность хранимых данных. Сказать что-либо про надежность хранения данных так же затруднительно. Однако с уверенностью можно сказать, что если вы потеряли информацию, хранимую в «облаке», то вы потеряли ее навсегда. Эксперты сходятся во мнении, что наиболее ценные для компании документы не стоит хранить на публичном «облаке» [12].

Таким образом, получается, что информация о надежности хранения данных в облаке неизвестна. Все зависит от используемого оборудования. О надежности хранения данных упоминается лишь с точки зрения защиты информации: защиты от «недобросовестных» сотрудников.

Персональные компьютеры

Надёжность персональных компьютеров обеспечивается технологическим процессом, т.е. какой уровень надежности был задан, такой надежность и будет.

Системы реального времени

В системах реального времени точность поступающей информации очень важна, ведь в соответствии с ней, будут приниматься оперативные решения. Здесь, как нигде, значение надежности хранения поступающей информации должно быть высоким.

Области применения систем хранения

Пользователи систем хранения данных используют технологии обеспечения надежности хранения данных, это означает, что для них надежности обеспечения хранения данных недостаточно. Это характерно для серверных систем, для систем реального времени этот вопрос стоит очень остро. Системы выбора носителей с указанной надежностью не существует, и пользователи должны выбирать носитель с предоставляемой производителями надежностью. Надежность и объем возрастают, а пользователь не может регулировать надёжность. Требования к надежности постоянно растут.

Теоретические аспекты надёжности передачи, хранения информации

В течение последних двух десятилетий несколько важных достижений стимулировали быстрое развитие кодов, исправляющих ошибки. Тот факт, что стоимость электронных устройств на интегральных схемах убывает с катастрофической скоростью, как и их размеры, ускорил развитие цифровой вычислительной техники и периферийных устройств вычислительных машин, что в свою очередь привело к драматическому росту количества данных, передаваемых от одной машины к другой. Недопустимость ошибок в вычислительных системах, а в некоторых случаях и критическая природа самих данных требовали использования оборудования, исключающего ошибки, или какого-либо рода кодов, исправляющих или обнаруживающих ошибки на выходе конечных устройств. В большинстве случаев более выгодным оказывается второй путь.

С другой стороны, значительные успехи были достигнуты и в самой области кодов, исправляющих ошибки. Создано несколько классов длинных мощных кодов. Кроме того, для

некоторых из этих классов кодов разработаны процедуры декодирования, которые не требуют большого количества оборудования.

Эти и другие достижения привели к тому, что в настоящее время использование кодов, исправляющих ошибки, стало действительно практически осуществимым в системах для передачи данных. Естественный прогноз состоит в том, что в ближайшем будущем указанные выше тенденции будут продолжать развиваться, и коды, исправляющие ошибки, будут находить все более широкое применение. Рассмотрим схему передачи информации.

Схема передачи информации

Принципиальная схема цифровой системы связи изображена на рис. 2. Основным отличием от традиционного представления схемы передачи информации является использование источника хранения (NAND флеш-память) информации в качестве канала. Источником информации является, как правило, сообщение, состоящее из двоичных или десятичных цифр, или же текст, записанный с помощью некоторого алфавита. Кодер преобразует эти сообщения в сигналы, которые записывается на запоминающее устройство. При использовании запоминающего устройства в качестве канала, искажение возникает за счёт выгорания транзисторов или какими-либо другими аппаратными сбоями. Затем искаженная информация считывается декодером, который восстанавливает записанное сообщение. Основная идея состоит в том, чтобы построить кодер и декодер, хотя она может включать в себя также задачу улучшения самого канала. Заметим, что кодер включает в себя устройство, которое обычно называют модулятором, а в декодер входит устройство, которое обычно называют детектором. Модулятор необходим для преобразования информационной последовательности в двоичный вид. Прицнип работы детектора противоположен принципу работы модулятора.

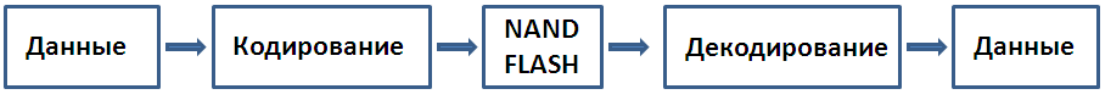


Рис. 2. Общая схема передачи информации

В большинстве случаев каналы связи, описываемые схемой, представленной на рис. 2, обладают определенной пропускной способностью для передачи информации, в случае использования запоминающего устройства – это максимальная скорость считывания/записи информации. Теория помехоустойчивого кодирования базируется на результатах исследований, проведенных Шенноном и сформулированных в виде теоремы: «При любой производительности источника сообщений, меньшей, чем пропускная способность канала, существует такой способ кодирования, который позволяет обеспечить передачу всей информации, создаваемой источником сообщений, со сколь угодно малой вероятностью ошибки» [13]. Обеспечение передачи информации с весьма малой вероятностью ошибки и достаточно высокой эффективностью возможно при кодировании чрезвычайно длинными последовательностями знаков. Кодирование с исправлением ошибок представляет собой метод обработки сообщений для повышения надёжности передачи по цифровым каналам.

Было доказано, что для соответствующим образом выбранных кодов и при любой заданной скорости передачи, число кодовых слов и число возможных принятых последовательностей являются экспоненциально растущими функциями, поэтому при больших размерностях кодового слова практически невозможно осуществить хранение в памяти кодера всех кодовых слов и хранение в памяти декодера способа отображения принятых последовательностей в сообщениях [14].

Помехоустойчивое кодирование

На первый взгляд помехоустойчивое кодирование реализуется весьма просто. В память кодирующего устройства (кодера) записываются разрешенные кодовые комбинации выбранного кода и правило, по которому с каждым из x сообщений источника сопоставляется одна из таких комбинаций y. Данное правило известно и декодеру [14].

Получив от источника определенное сообщение, кодер отыскивает соответствующую ему комбинацию и посылает ее в канал или записывает на запоминающее устройство. В свою очередь, декодер, приняв комбинацию, искаженную помехами, сравнивает ее со всеми y комбинациями списка и отыскивает ту из них, которая ближе остальных к принятой (согласно выбранной метрике). В связи с ограничением на память, применение табличных методов кодирования и декодирования при достаточно длинных кодах не эффективно и технически невозможно.

Поэтому основное направление теории помехоустойчивого кодирования заключается в поисках таких классов кодов, для которых кодирование и декодирование осуществляются не перебором

таблицы, а с помощью некоторых регулярных правил, определенных алгебраической структурой кодовых комбинаций [14].

Линейные коды

Под линейными понимаются такие двоичные коды, в которых множество всех разрешенных блоков является линейным пространством относительно операции поразрядного сложения по модулю 2.

Множество линейных комбинаций образует линейное пространство, содержащее 2^k блоков, т.е. линейный код, содержащий 2^k блоков длиной n, обозначают *(n, k)*. При заданных n и k существует много различных *(n, k)*-кодов с различными кодовыми расстояниями d, определяемых различными порождающими матрицами.

Описание кода

Кодер для линейных блочных кодов делит информационную последовательность (последовательность бит) на блоки длиной k символов. Кодер преобразует информационные блоки X в более длинные последовательности Y, состоящие из n символов, называемых допустимыми словами (кодовые слова). Добавленные кодером к каждому блоку символы называются избыточными (проверочные).

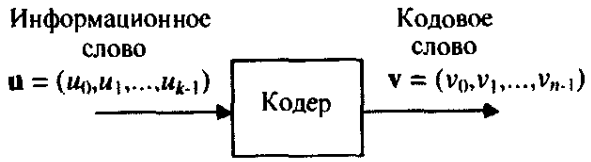


Рис. 3. Блок кодирования

Оператор кодирования придаёт системе свойство избыточности, причём избыточные символы находятся в тесной взаимосвязи с кодируемыми информационными символами, т.е. данная информация перераспределяется и на избыточные символы.

Одним из существенных качеств кода является систематичность. Систематический код имеет неизменную длину, то есть постоянную длину информационной и избыточной частей кода.

Порождающая матрица

Рассмотрим основные способы описания кода. Простейшим способом описания корректирующих кодов является табличный способ, при котором каждому блоку ставится в соответствие кодовое слово из таблицы кода. Существует степенная зависимость между размером блоков и размерностью таблицы, что делает неудобным использование таблицы. Наиболее наглядным способом описания блочных кодов является их задание с помощью порождающей матрицы (1).

Порождающая матрица G для *(S, L)*-кода с проверкой на четность представляет собой двоичную матрицу с размерами L на N [14].

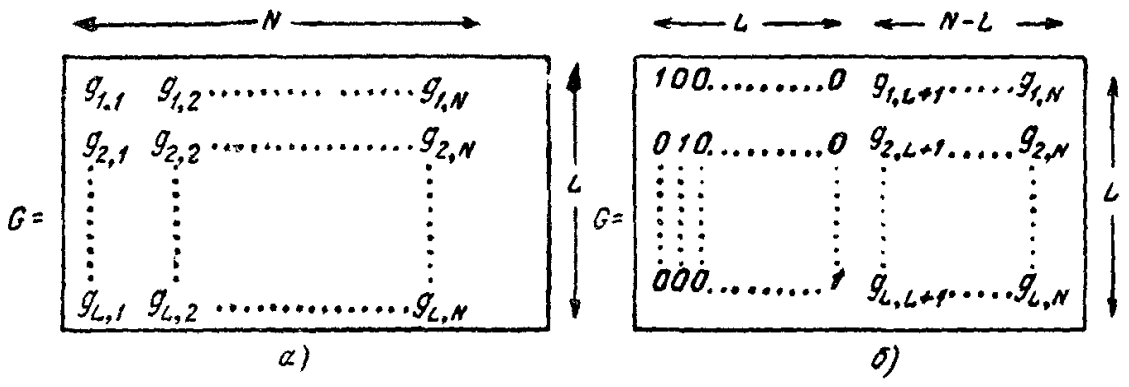


Рис. 4.

а) произвольный код с проверкой на чётность;

б) систематический код с проверкой на чётность

Для каждого блока *x* ставится в соответствие кодовое слово *y,* рассчитываемое по формуле:

$$y = x \bullet G, \quad (1)$$

где $x \bullet G$ представляет собой матричное произведение, использующее сложение по модулю 2, $g_1, g_2, \ldots g_N$ $g_1, g_2, \ldots g_N$ – некоторый базис кода *y(L,N)*.

Из свойств базиса непосредственно следует, что любой кодовый вектор может быть представлен в виде линейной комбинации строк $g_1, \ldots g_N$ матрицы *G* и, наоборот, что любая линейная комбинация строк $g_1, \ldots g_N$ $g_1, \ldots g_N$ матрицы *G* представляет собой кодовый вектор и, более того, различные линейные комбинации задают различные кодовые векторы.

Важной характеристикой матрицы LDPC-кода является отсутствие циклов определённого размера. Под циклом длины 4 понимают образование в проверочной матрице прямоугольника, в углах которого стоят единицы. Для этого достаточно, чтобы любые два столбца проверочной матрицы LDPC кода не имели более одной общей ненулевой позиции [15]. Циклы большей длины можно определить, если в проверочной матрице построить граф, вершинами которого являются единицы, а рёбра – все возможные соединения вершин, параллельные сторонам матрицы (то есть вертикальные или горизонтальные линии). Минимальный цикл в этом графе и будет минимальным циклом в проверочной матрице LDPC-кода.

Проверочная матрица

Любой систематический код может быть определён проверочной матрицей H*H*, обладающей определёнными свойствами. Если некоторая последовательность *y* является кодовым словом, то выполняется следующее условие:

$$y \cdot H^T = 0 \; y \bullet H^T = 0, \quad (2)$$

т.е. проверочная матрица *H* ортогональна любой кодовой последовательности данного кода.

Синдром и обнаружение ошибки линейным блочным кодом

Пусть $x = (x_1, x_2, \ldots, x_n)$ $x = (x_1, x_2, \ldots, x_n)$ – кодовое слово, переданное по каналу с помехами;

$y = (y_1, y_2, \ldots, y_n)$ $y = (y_1, y_2, \ldots, y_n)$ – принятая последовательность, которая в силу влияния помех может отличаться от переданной. Для описания возникающих в канале ошибок используется вектор ошибок $e = (e_1, e_2, \ldots, e_n)$ $e = (e_1, e_2, \ldots, e_n)$, который представляет собой двоичную последовательность длиной *n* с единицами в тех позициях, в которых произошла ошибка.

Например, вектор ошибок *e = (0001000)* означает однократную ошибку в четвертом бите, *(1100000)* — двукратную ошибку в первом и втором битах.

При передаче кодового слова *x* по каналу с шумом принятая последовательность будет иметь вид:

$$y = x + e, \quad (3)$$

где *x* – переданное кодовое слово; *e* – вектор, описывающий ошибки в канале.

Например, *x* = (0001000), *e* = (0001000). Тогда *y* = (0000000).

Приняв вектор *y*, декодер сначала должен определить, имеются ли в принятой последовательности ошибки, и, если да, то должны выполниться действия но их исправлению. Чтобы проверить наличие ошибок, декодер вычисляет следующую *(n – k)*-последовательность:

$$S = (S_1, S_2, ..., S_{n-k}) = y \times H^T,$$

где *y* – принятая кодированная последовательность; $H^T H^T$ – проверочная матрица данного кода. При этом *y* является кодовым словом тогда, когда *S = (000... 0)* и не является кодовым словом данного кода, если $S \neq 0$. Последовательность *S* служит признаком наличия ошибок в принятой последовательности и называется синдромом принятого вектора *y*. Некоторые сочетания ошибок невозможно обнаружить, используя синдром. Например, если переданное кодовое слово *x* под влиянием помех превратилось в другое кодовое слово этого же кода, тогда синдром:

$$S = y \times H^T = (0,0...0) \quad (4)$$

и декодер ошибки не обнаружит.

Преимуществом линейных, в частности систематических, кодов является то, что в кодере и декодере не нужно хранить большие таблицы всех кодовых комбинаций, а при декодировании не нужно производить большое количество сравнений [14].

Однако, для получения высокой верности связи следует применять коды достаточно большой длины. Применение систематического кода в общем случае, хотя и позволяет упростить декодирование по сравнению с табличным способом, все же при значениях *n* порядка нескольких десятков не решает задачу практической реализации.

Циклические коды

Был предложен ряд кодов и способов декодирования, при которых сложность декодера растет не экспоненциально, а лишь как некоторая степень *n*. Циклические коды просты в реализации и при невысокой избыточности обладают хорошими свойствами обнаружения ошибок. Название циклических кодов связано с тем, что каждая кодовая комбинация, получаемая путем циклической перестановки символов, также принадлежит коду. Так, например, циклические перестановки комбинации 1000101 будут также кодовыми комбинациями 0001011, 0010110, 0101100 и т.д.

Представление кодовых комбинаций в виде многочленов *F(x)* позволяет установить однозначное соответствие между ними и свести действия над комбинациями к действию над многочленами. Сложение двоичных многочленов сводится к сложению по модулю 2 коэффициентов при равных степенях переменной *x*. Умножение производится по обычному правилу умножения степенных функций, однако полученные коэффициенты при данной степени складываются по модулю 2. Деление осуществляется, как обычное деление многочленов, при этом операция вычитания заменяется операцией сложения по модулю 2. Циклическая перестановка кодовой комбинации эквивалентна умножению полинома *F(x)* на *x* с заменой на единицу переменной со степенью, превышающую степень полинома [14].

Любой полином *G(x)* степени $r < n$, который делит без остатка двучлен $x^n - 1$, может быть *порождающим полиномом* циклического *(n, k)*-кода, где $k = n - r$. В этот код входят те полиномы, которые без остатка делятся на *G(x)*.

Особую роль в теории циклических кодов играют *неприводимые многочлены G(x),* т.е. полиномы, которые не могут быть представлены в виде произведения многочленов низших степеней.

Идея построения циклического кода *(n,k)* сводится к тому, что полином *Q(x)*, представляющий информационную часть кодовой комбинации, нужно преобразовать в полином *F(x)* степени не более $n - 1$, который без остатка делится на порождающий полином *G(x)* (неприводимый многочлен) степени $r = n - k$. Рассмотрим основные шаги построения циклического кода:

- представляем информационную часть кодовой комбинации длиной *k* в виде полинома *Q(x)*;
- умножаем *Q(x)* на одночлен x^r и получаем $Q(x)x^r$;
- делим полином $Q(x)x^r$ на порождающий полином *G(x)* степени *r*, при этом получаем частное от деления *C(x)* такой же степени, что и *Q(x)*:

$$\frac{Q(x)x^r}{G(x)} = C(x) \oplus \frac{R(x)}{G(x)}, \qquad \frac{Q(x)x^r}{G(x)} = C(x) \oplus \frac{R(x)}{G(x)},$$

(5)

где $R(x)$ – остаток от деления $Q(x)x^r$ на $G(x)$;

- умножив обе части на $G(x)$, получим $F(x) = C(x)G(x) = Q(x)x^r \oplus R(x)$. Полином $F(x)$ делится $F(x) = C(x)G(x) = Q(x)x^r \oplus R(x)$

без остатка на $G(x)$, т.е. представляет собой разрешенную комбинацию циклического (n, k) -кода.

Таким образом, разрешенную кодовую комбинацию циклического кода можно получить двумя способами: умножением кодовой комбинации простого кода $C(x)$ на полином $G(x)$ или умножением кодовой комбинации $Q(x)$ простого кода на одночлен x^r и добавлением к этому произведению остатка $R(x)$.

Коды с проверкой на чётность

Код с проверкой на четность является частным случаем отображения двоичной последовательности длины L в двоичную последовательность некоторой большей длины N [14]. Рассмотрим довольно простой, но широко используемый пример проверки на четность. Допустим, что последовательность двоичных символов кодируется с помощью простого добавления одного двоичного символа в конце последовательности; этот последний символ выбирается таким образом, чтобы общее число единиц в кодовой последовательности было четным. Легко видеть, что если затем изменить один из символов последовательности, то общее число единиц станет нечетным, что обнаруживает факт наступления ошибки. Однако, если произойдут две ошибки, число единиц вновь станет четным и ошибки не будут обнаружены. Кодирование такого типа широко используется при записи на магнитные ленты, а также и в других случаях, когда желательна некоторая небольшая возможность обнаружения ошибок при минимальных затратах на оборудование.

Одну из причин рассмотрения кодов с проверкой на четность как систематических, так и несистематических, можно понять, если рассмотреть реализации кодера. Для кодера с проверкой на четность необходимы регистр для запоминания последовательности сообщения и, регистр для запоминания кодового слова y и сумматоры по модулю 2, число которых пропорционально NL. Поэтому использование кодеров с проверкой на четность позволяет избежать экспоненциального роста объема памяти с ростом L, неизбежного при использовании произвольного блокового кода с $2L$ кодовыми словами, структура которого не выбиралась специальным образом [14].

LDPC-коды

Коды с малой плотностью проверок на четность (LDPC-код от англ. Low-density parity-check code, LDPC-code, низкоплотностный код) были впервые предложены Р. Галлагером и позднее исследовались во многих научных работах.

LDPC-коды представляют собой линейные блоковые коды, задаваемые с помощью низкоплотностной порождающей матрицы H, характеризуемой относительно малым числом единиц в строках и столбцах, порождающей матрице кода ставится в соответствие граф Таннера, в котором для представления строк и столбцов используются определенным образом связанные между собой узлы. На рисунке 5 представлен граф Таннера для кода с порождающей матрицей H.

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

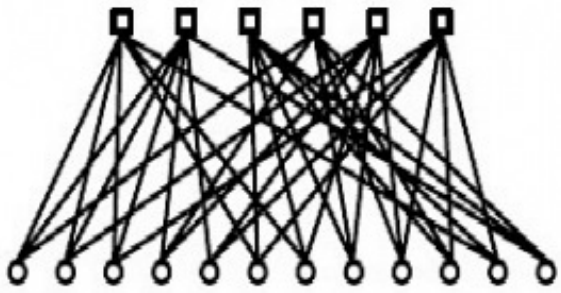


Рис. 5. Порождающая матрица и граф Таннера LDPC-кода

Для построения порождающей матрицы используются два принципа: первый основан на генерации начальной порождающей матрицы с помощью псевдослучайного генератора, второй – использование специальных методов, основанных, например, на группах и конечных полях. Коды, полученные этими способами, называют структурированными. Лучшие результаты по исправлению ошибок показывают именно случайные коды, однако структурированные коды позволяют использовать методы оптимизации процедур хранения, кодирования и декодирования, а также получать коды с более предсказуемыми характеристиками.

Кодирование и декодирование

Как и для любого другого линейного кода, для декодирования используется свойство ортогональности порождающей и транспонированной проверочной матриц:

$$\mathbf{H} \cdot \mathbf{M}^{\mathbf{T}} = \mathbf{0}\, H \cdot M^T = 0, \quad (6)$$

где H – порождающая матрица, и M – проверочная. Тогда для каждого принятого без ошибок кодового слова y выполняется отношение:

$$\mathbf{y} \cdot \mathbf{M}^{\mathbf{T}} = \mathbf{0}\, y \cdot M^T = 0, \quad (7)$$

а для принятого кодового слова с ошибкой:

$$\mathbf{y} \cdot \mathbf{M}^{\mathbf{T}} \neq \mathbf{0}\, y \cdot M^T \neq 0, \quad (8)$$

где y – принятый вектор.

В случае, если после умножения принятого кодового слова на транспонированную проверочную матрицу получается ноль, блок считается принятым без ошибок.

Декодирование по максимуму правдоподобия кода обозначает нахождение по заданному принятому вектору y такого кодового слова из множества допустимых значений кодовых слов, которое максимизирует вероятность того, что передавалось именно эта информация при условии принятия вектора y. Первоначально информация кодируется с помощью графа Таннера или порождающей матрицы для получения множества допустимых значений. При считывании информация сравнивается с элементами множества допустимых значений. Мерой схожести является Евклидово расстояние. Если считанный код не является одним из допустимых, очевидно, что информация считана с ошибкой. При минимальном расстоянии до одного из допустимых значений мы можем утверждать, что считанная информация тождественна выбранному допустимому значению согласно критерию минимального расстояния.

Для того, чтобы оценить качество работы декодеров обычно рассматривается оценка вероятности ошибки декодирования на информационный бит, рассчитываемая как количество

ошибочных бит после декодирования к общему количеству передаваемых бит. Итеративные схемы декодирования кодов с низкой плотностью проверок на четность имеет существенное различие от декодеров, использующих принцип максимального правдоподобия. Суть итеративного декодирования заключается в том, что нахождения кодового слова производится за несколько итераций, последовательно уточняя результат. Приведём основные схемы декодирования.

Жесткое декодирование

Схема «жесткое» декодирование используется в случае небольшого количества ошибок при считывании. Для проверки будем брать строки из проверочной матрицы $\boldsymbol{h = \{h_0, \ldots, h_{N-1}\}}$ $h = \{h_0,\ldots,h_{N-1}\}$. При удачной проверке скалярное произведение считанного

$$\boldsymbol{y = \{y_0, \ldots, y_{N-1}\}}$$

вектора $y = \{y_0,\ldots,y_{N-1}\}$ на любую строку из проверочной матрицы даёт ноль. Будем говорить, что элемент y_i считанного вектора y участвует в проверке $\boldsymbol{h = \{h_0, \ldots, h_{N-1}\}}$ $h = \{h_0,\ldots,h_{N-1}\}$ тогда, когда соответствующий элемент проверки h_i не равен нулю [22].

Итерация жесткого декодирования инвертированием битов принимает следующий вид:

- Для считанного вектора просчитываются все проверки.
- Если в более чем половине не выполнившихся проверок участвовал один и тот же бит, то бит инвертируется.
- После анализа всех символов считанный вектор проверяется на принадлежность коду.
- В случае, если вектор является кодовым словом, декодирование заканчивается, иначе переходим к следующей итерации алгоритма.

Такой вид декодирования используется только для кодов с низкой плотность проверок на чётность потому, что данный вид кода характеризуется малым число ошибок и тогда невыполнения большого количества проверок для бита считанного вектора будет означать наличие в нём ошибки.

Декодирование по вероятностям

Декодирование по вероятностям относится к «мягкому» декодированию. Под «мягким» декодированием обычно понимается декодирование на основе вектора, состоящего из вещественных чисел, получаемых путём пересчёта вероятностей. Для этого формируются два вектора вероятностей (двоичный случай) того, что в считанном векторе на данной позиции находился заданный символ. Для каждого ненулевого элемента проверочной матрицы ставится в соответствие две величины: $q_{i,j}^x$ и $r_{i,j}^x$. Под величиной $q_{i,j}^x$ понимается вероятность того, что *j*-й символ считанного символа будет равен *x,* согласно всем проверкам, кроме *i*-й. Величина $r_{i,j}^x$ подразумевает под собой вероятность того, что проверка *i* успешна при условии, что *j*-й символ считанного вектора *x*, а все остальные символы проверок имеют распределение вероятностей, заданное величинами:

$\{q_{i,j}^x : j \text{ из } N(i) \setminus j\}$,

где $N(i)$ – множество символов, входящих в *i*-ю проверку [22].

Изначально для работы алгоритму необходимо произвести инициализацию, затем алгоритм пересчитывает вероятность символа считанного вектора, применяя правило Байеса для апостериорной вероятности события.

Итерация разбивается на следующие шаги:

- Для всех проверок вычисляются величины $\Delta r_{i,j}$ и пересчитываются вероятности $r_{i,j}^x$ для *x = {0, 1}*.
- Затем для каждого символа считанного вектора пересчитываются вероятности $q_{i,j}^x$.
- Происходит формирование векторов псевдо апостериорной вероятности q_{0j} и $q_{1,j}$
- На последнем шаге формируется вектор решения y' согласно следующему правилу: $y' \bullet j = 1$, если $jq_{1,j} > \frac{1}{2}$, иначе 0. Если вектор y' является кодовым словом, декодирование заканчивается, в противном случае переходим к следующей итерации алгоритма.

Сложность данного алгоритма выше, чем у «жесткого» декодирования, при этом повышается качество декодирования за счёт использования дополнительной информации на выходе канала. Существует прямая зависимость точности данного алгоритма от инициализации, чем точнее она произведена, тем точнее будет конечный результат.

Быстрое декодирование

Декодирование пересчётом вероятностей является довольно эффективным методом для каналов с непрерывным выходом, однако, исходя из того, что сложность его значительно выше, чем у «жесткого» декодирования, это создает предпосылки для поиска оптимальных алгоритмов декодирования.

Набольшей популярностью среди алгоритмов быстрого декодирования имеет алгоритм «min-sum», являющийся упрощением декодера «belief propagation», а также алгоритм UMP (Uniformly Most Powerful).

Сложность декодера UMP значительно ниже по сравнению с декодером, пересчитывающим вероятности из-за того, что пересчет надежностей выполняется по упрощенной схеме (в качестве «весов» используется надежность проверок), а также за счет возможности использования

исключительно целочисленных операций сложения и сложения по модулю два. Также можно выделить тот факт, что декодеру не требуется знать характеристики шума в канале (дисперсию и т. д.), следовательно, такой декодер является более универсальным.

Многопороговое декодирование

Основная идея многопорогового декодирования по надежностям состоит в изменении значения порогов инвертирования символов от одной итерации к другой следующим образом: на первых итерациях порог инвертирования символов выбирается так, чтобы количество инвертированных символов было минимальным (вплоть до инвертирования только одного символа на первой итерации); на последующих итерациях пороги инвертирования постепенно повышаются.

При многопороговом декодировании, если на первой итерации была исправлена хотя бы одна ошибка, декодирование на последующих итерациях становится значительно проще и общее качество декодирования улучшается. Для работы декодеру необходимо задать лишь надежности.

Декодер, работающий по многопороговой схеме, позволяет получить вероятность ошибки декодирования на 0,1 – 0,4 дБ лучше по сравнению с быстрым декодером по надежностям UMP, практически приближаясь к вероятности ошибки, получаемой при вероятностном декодировании кодов с низкой плотностью проверок на четность [22]. Многопороговый декодер обладает универсальностью и применимостью для любой конструкции таких кодов.

Стоит заметить, что нерегулярные LDPC-коды оказываются эффективнее, чем регулярные, так как в нерегулярных кодах из-за различного числа единиц в строках и столбцах информационные символы защищены по-разному. Результатом чего является более быстрое декодирование защищённых бит, которые «помогают» при декодировании менее защищённых бит.

Рассмотрение LDPC на примере LDPC кода (3,6)

Линейные коды получаются в результате перемножения сообщения *x* на порождающую матрицу *G*[IP]. Каждой порождающей матрице ставится в соответствие матрица проверки четности. Эта матрица позволяет исправлять ошибки в полученных сообщениях путем вычисления синдрома.

Построим начальную порождающую матрицу для описания кода случайным образом с заданным значением плотности:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

(9)

Матрица проверки четности находится из матрицы идентичности *I* и транспонированной матрицы *P*:

$$H(P^T I) = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

В силу того, что мы разбиваем информационную последовательность на блоки длиной 3 бита, то мы имеем 2^3 различных вариаций блоков. Для всех возможных блоков построим кодовые слова, как произведение считанного блока на порождающую матрицу (1). В результате к каждому блоку будут приписаны 3 избыточных бита. Данное множество кодовых слов образует множество допустимых значений (см. рис. 6).

Построим с помощью порождающей матрицы множество кодовых слов(6):

Блоки	Кодовые слова
000	00000
001	00110
010	01000
011	01110
100	10010
101	10100
111	11100

Рис. 6. Соответствие блоков и кодовых слов

Предположим, что мы считали блок $x = (011)$, тогда кодовое слово согласно формуле (6):

$$y = [001]\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} = [011011].$$

Допустим, что при чтении исказился последний бит кодового слова, и мы считали$y' = [011010]$ $y' = [011010]$.

Подсчитаем синдром для этого полученного кодового слова по формуле 4:

$$[011010]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [001] S = [011010]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [001].$$

При этом y является кодовым словом, если синдром S = (000... 0), в случае$S \neq 0$ $S \neq 0$ в принятой последовательности допущена ошибка. В нашем случае $S \neq 0$$S \neq 0$, следственно принятый блок содержит ошибку. Некоторые сочетания ошибок невозможно обнаружить, используя синдром. Например, если переданное кодовое слово x под влиянием помех превратилось в другое кодовое слово этого же кода.

Каждое значение вектора синдрома является декартовым произведением полученного кодового слова на матрицу проверки на чётность. Исследуем, действительно ли синдром при одной ошибке является столбцом проверочной матрицы. В таблице 1 взаимопоставлены значение синдрома для кодового слова с ошибкой, получаемого путём сложения кодового слова y с ошибочным вектором посредством операции XOR.

Лёгко заметить, что если было искажение в первом бите кодового слова, то синдром совпадает с первым столбцом проверочной матрицы, т.е. значение синдрома при одной ошибке совпадает всегда с одним из столбцов проверочной матрицы.

Таблица 2.Синдром для кодовых слов с ошибками

Ошибки					Кодовое слово							Синдром		

Пример жёсткого декодирования

«Жёсткое» декодирование – это одна из самых простых схем декодирования для двоичного симметричного канала при небольшом количестве ошибок в канале. Реализация алгоритма жёсткого декодирования в Excel продемонстрирована на рис. 7.

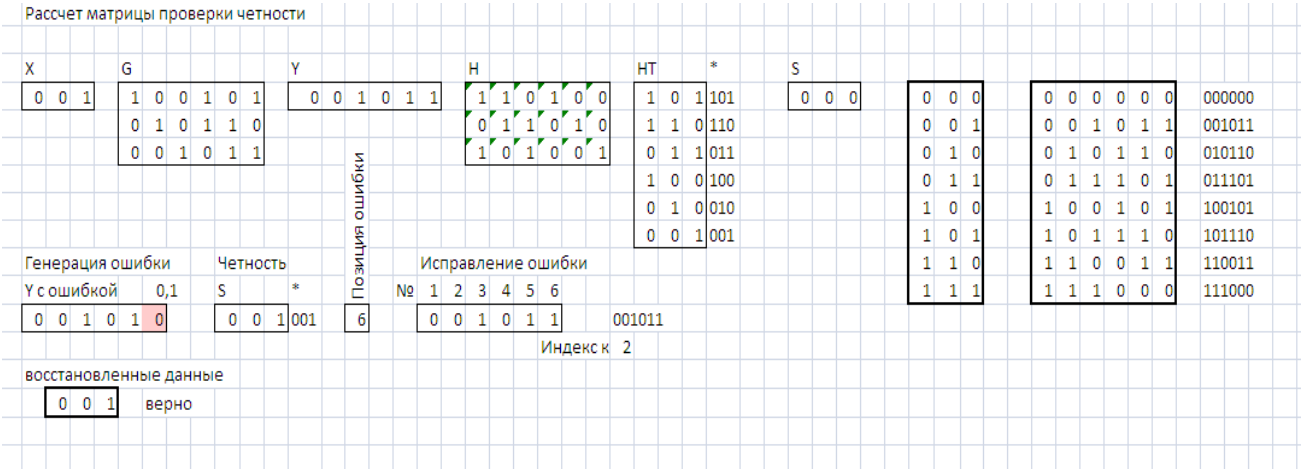


Рис. 7. Реализация жесткого декодирования в Excel

Алгоритм жеёсткого декодирования принимает следующий вид:

На первом этапе вычисляем синдром для считанного кодового слова. В случае, если все проверки дали 0, т.е. синдром $S = 0$, то искажения при считывании не произошло и мы имеем дело с кодовым словом, которое принадлежит множеству допустимых кодовых слов. Если $S \neq 0$ переходим к шагу 2.

- На данном шаге находится индекс столбца проверочной матрицы, который тождественен синдрому.
- Инвертируем в искажённом кодовом слове бит, стоящий на полученном, на шаге 2 индексе. Далее считанный вектор проверяется на принадлежность коду. Если вектор является кодовым словом, декодирование заканчивается, в противном случае выполняется следующая итерация алгоритма.
- Применение данной процедуры декодирования для кодов с низкой плотностью проверок на четность вполне обосновано, так как большинство проверок будут содержать одну ошибку или не будут содержать ошибок вообще и тогда невыполнение большого количества проверок для символа принятого слова будет обозначать наличие в нем ошибки.

Декодирование принципом максимального правдоподобия

Необходимо проанализировать влияние вида порождающей матрицы алгоритма на способность алгоритма к нахождению и исправлению ошибки по принципу максимального правдоподобия.

Одним из главных параметров, влияющих на вероятность ошибочного декодирования, является кодовое минимальное расстояние [16]. Таким образом, чем больше кодовое расстояние, тем меньше вероятность ошибочного декодирования.

Матрица кодирования должна обладать определённым свойством: разброс допустимых значений кодовых слов, получаемых после применения операции (6), должен быть максимален, то есть расстояние (согласно выбранной метрики) между допустимыми значениями принимало максимальную величину.

Построим порождающую матрицу случайным образом с заданным значением плотности.

Для первоначального анализа матрицы кодирования используется распределение допустимых кодов во множестве всевозможных кодов, чем дальше друг от друга допустимые коды разбросаны, тем выше вероятность декодирования при искажении, то есть критерием оценки качества построенной порождающей матрицы является расстояние между допустимыми значениями. Под расстоянием понимается евклидово расстояние.

Смоделируем разброс допустимых значений от вида порождающей матрицы. Вид порождающей матрицы напрямую зависит от значения плотности. В результате можно сделать выпод, что разброс допустимых значений кодовых слов в большинстве случаев принимает максимальную величину при плотности матрицы равной 0.35 (см. Таблицу 3).

Таблица 3. Зависимость плотности матрицы от суммарного расстояния

Суммарное расстояние между допустимыми значениями	Плотность матрицы
37,363	0,15
41,756	0,20
48,01	0,25
48,17	0,30
52,51	0,35
50,34	0,40
49,70	0,50

Согласно декодированию по принципу максимального правдоподобия для проверки кодового слова на предмет наличия ошибки, мы должны сначала подсчитать синдром. Если синдром принимает нулевую величину (S = (000... 0), то ошибки искажения при чтении кодового слова не произошло, иначе необходимо детектировать ошибку. Допустим при чтении информации произошло n ошибок. Для оценки возможности декодирования рассчитаем евклидово расстояния от считанного кодового слова до всех допустимых значений. В случае, если минимальное расстояние достигается только до одного из допустимых значений и при этом расстояние до остальных не соизмеримо с ним, то мы можем смело говорить, что исправление ошибки возможно.

Рассмотрим предыдущий пример. Допустим, что мы считали блок $x = (011)$, тогда кодовое слово согласно формуле (6):

$$y = [001]\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} = [011011].$$

Допустим, что при чтении произошла ошибка и последний бит кодового слова исказился, и мы считали $y' = [011010]$ $y' = [011010]$ вместо правильного кодового слова $y = [011011]$

$y = [011011]$. Синдром для кодового слова не принимает нулевое значение$S = [001] \neq 0$ $S = [001] \neq 0$. Тем самым мы детектировали ошибку. Попробуем исправить ошибку по принципу максимального правдоподобия. Найдём евклидово расстояние между полученным кодовым словом и всеми кодовыми словами из множества допустимых значений:

Таблица 4. Расстояние кодового слова до допустимых кодовых слов

Возможные кодовые слова						Расстояние
0						1,414213562
0						1,732050808
0						1,414213562
0						1
1						2
1						2,236067977
1						2
1						1,732050808

Проанализировав таблицу 4, можно заметить считанного кодового слово наиболее близко находится к кодовому слову $y_4 = [011011] y_4 = [011011]$. За это кодовое слово мы и принимаем наше искажённое кодовое слово. Кодовому слово можно сопоставить блок закодированной информации. Иногда случаются такие ситуации, что мы не можем однозначно трактовать к какому кодовому слову отнести считанное искажённое кодовое слово. Это происходит в случае, когда минимальное расстояние до двух или более кодовых слов, в данном случае декодирование по данному принципу невозможно.

Турбо-код

Турбо-коды представляют собой сравнительно новый тип кодов для исправления ошибок, возникающих при передаче цифровой информации по каналам связи с шумами. Турбо-код обладает уникальной способностью помехоустойчивости передачи информации, близкие к теоретически достижимым значениям при умеренной сложности реализации кодеров.

Разработка турбо-кодов развивается по двум направлениям: сверточные турбо-коды, образованные путем параллельного соединения двух или более сверточных кодеров, и блочные турбо-коды, образованные путем последовательного соединения двух или более блочных кодеров. При высоких кодовых скоростях блочные коды более эффективны в использовании.

Рассмотрим принцип построения турбо-кодов на примере двухмерного блочного турбо-кода.

Двухмерный блочный турбо-код изображается в виде прямоугольника и основан на двух систематических кодах: горизонтальных$C_x = (n_x; k_x)$ $C_x = (n_x; k_x)$ и вертикальных $C_y = (n_y; k_y)$ $C_y = (n_y; k_y)$. Общая информационная емкость кода $k = k_x \cdot k_y$ $k = k_x \cdot k_y$, длина $n = n_x \cdot n_y$ $n = n_x \cdot n_y$ [17].

Считанный поток битов, поступающих на кодер, построчно записывается в матрицу. Сначала кодируется k_y k_y строк, затем n_x n_x столбцов, в результате чего получается кодированная матрица, содержащая как информационные, так и проверочные избыточные символы. Каждая строка матрицы

представляет собой кодовое слово $C_x = (n_x; k_x) C_x = (n_x; k_x)$ и состоит из k_x информационных символов и $n_x - k_x$ проверочных. Каждый столбец, соответственно, представляет собой кодовое слово C_y C_y и состоит из k_y информационных символов и $n_y - k_y$ проверочных. Как правило, передача бит из кодированной матрицы в последующие цепи осуществляется построчно.

Достоинство турбо-кодов состоит в том, что они допускают итеративную процедуру декодирования. Итеративный декодер двумерного блочного турбо-кода представляет собой последовательное соединение двух элементарных декодеров и основан на вычислении апостериорных вероятностей двоичных символов кодовых слов C_x и C_y. Каждый из элементарных декодеров выносит решение о переданном символе на основе критерия максимальной апостериорной вероятности, чем обеспечивается минимум вероятности ошибочного декодирования каждым элементарным декодером [18].

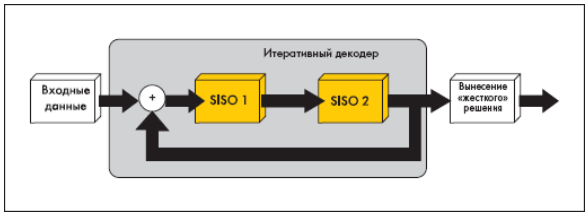


Рис. 8. Блок-схема итеративного декодера

На первой итерации от демодулятора на вход первого декодера поступают оценки («мягкие» решения) символов. На выходе первого декодера формируется «мягкое» решение символов, которое затем используется в качестве входной информации для второго декодера.

На второй и последующих итерациях декодирования входные данные обновляются и используются как априорная информация о переданных символах для первого декодера. Процедура повторяется от итерации к итерации, увеличивая вероятность правильного декодирования. Окончание процесса декодирования происходит после выполнения заданного количества итерационных циклов. В этом случае после последней итерации от «мягких» решений берется знаковый разряд, который и является выходом итеративного декодера.

Таким образом, на вход элементарных декодеров поступают «мягкие» решения, результат декодирования на выходе элементарных декодеров – также «мягкое» решение. По этой причине такие схемы получили название декодеров с мягким входом и мягким выходом (Soft Input Soft Output – SISO). Изложенный алгоритм декодирования обладает высокой эффективностью, так как каждая последующая итерация увеличивает достоверность декодирования, а количество итераций ограничено допустимой задержкой на декодирование.

Важным преимуществом турбо-кодов является независимость сложности декодирования от длины информационного блока, что позволяет снизить вероятность ошибки декодирования путём увеличения его длины. Однако у турбо-кода и есть ряд недостатков [13].

Один важный недостаток турбо-кодов – сравнительно небольшое кодовое расстояние (то есть минимальное расстояние между двумя кодовыми словами в смысле выбранной метрики). Это приводит к тому, что, хотя при большой входной вероятности ошибки (то есть в плохом канале) эффективность турбо-кода высока, при малой входной вероятности ошибки эффективность турбо-кода крайне ограничена. Поэтому в хороших каналах для дальнейшего уменьшения вероятности ошибки применяют не турбо-коды, а LDPC-коды.

Коды Рида-Соломона (RS)

Совершенным кодом, нашедшим применения в современной видеозаписи, является код Рида-Соломона, требующий добавления избыточных символов в расчете двух проверочных символов на одну исправляемую ошибку. Коды Рида-Соломона широко применяются в цифровой технике. Они обладают определенными оптимальными свойствами и для них разработаны относительно простые и конструктивные методы кодирования. Коды Рида-Соломона не являются двоичными. Стоит обратить внимание на то, что символами кодовых слов являются не двоичные знаки, а элементы множества

чисел, состоящего более чем из двух знаков (каждый символ может заменяться соответствующей двоичной комбинацией).

Коды Рида-Соломона относятся к классу циклических кодов. Кодирование и декодирование, обнаруживающее и исправляющее ошибки, – это вычислительные процедуры, которые для циклических кодов удобно выполнять как действия с многочленами и которые допускают относительно простую схемотехническую реализацию в виде цифровых автоматов на базе регистров с обратными связями. Если кодируемая информация X описывается набором из k символов (размер блока), то этому набору или слову ставится в соответствие информационный многочлен $I(x)$.

Так как цель кодирования – исправление ошибок, достигаемое благодаря введению избыточности, то каждому информационному слову должно соответствовать кодовое слово y большей длины с добавленной избыточностью в виде дополнительных символов. Кодовое слово также можно записать в виде многочлена $C(x)$. В соответствии с теорией циклических кодов образование кодовых слов производится с помощью порождающего многочлена $G(x)$, степень которого равна числу дополнительных (проверочных) символов [20].

Процедура нахождения кодового слова заключается в умножении информационного многочлена на порождающий многочлен кода. Таким образом, все кодовые слова получены умножением разных информационных многочленов на один и тот же порождающий многочлен. Следовательно, они все делятся на порождающий многочлен без остатка. Это обстоятельство дает ключ к декодированию, позволяющему обнаруживать и исправлять ошибки, возникшие при передаче данных. Слова, которые не делятся без остатка на порождающий многочлен, не являются кодовыми и, следовательно, содержат ошибки. Теория кодов Рида-Соломона показывает, что если цель кодирования – исправить t ошибок в кодовом слове, то степень порождающего многочлена $(n - k) = 2t$ [20]. Следовательно, кодовое слово должно содержать два дополнительных символа на одну исправляемую ошибку. Для создания нового кода с заданными свойствами надо образовать соответствующий порождающий многочлен.

Записываемый на магнитный носитель кодовый блок может претерпеть искажения, например, из-за выхода из строя транзисторов или шума. Это можно в общем виде описать добавлением к кодовому блоку набора ошибок *e(t)*, представляемого в виде многочлена. Найдя остаток от деления принятого многочлена на порождающий, можно понять, были ли на самом деле ошибки. Если остаток равен нулю, значит принятое слово является кодовым и ошибок не было. Если остаток не равен нулю, то при передаче были ошибки. Остаток от деления дает многочлен, зависящий только от многочлена ошибки. Его называют синдромным многочленом. Синдромный многочлен зависит только от конфигурации ошибок, т.е. является описанием ошибок. Если число ошибок не превышает заданный предел t, то между многочленом ошибки и синдромным многочленом существует однозначное соответствие и с помощью определенных вычислений можно найти коэффициенты многочлена ошибок по синдромному многочлену. Таким образом, можно восстановить переданный кодовый многочлен: как разность принятого многочлена и многочлена ошибок. Разделив рассчитанный многочлен на порождающий многочлен, можно найти информационный многочлен и тем самым закончить этап декодирования.

В реальной жизни при применении RS-кодирования функции, обычно характеризующие какое-либо из устройств, кодирования или декодирования, часто применяются в программируемой логической схеме. Функция, часто выполняемая после RS-кодирования – перемежение. Задача перемежителя состоит в том, чтобы зашифровать символы в нескольких RS-закодированных словах перед передачей, эффективно распределив ошибку в линии передачи пакетных данных по нескольким кодовым словам. Такое распределение по нескольким кодовым словам увеличивает шансы на возможность исправления ошибки [21].

БЧХ-коды

Коды Боуза-Чоудхури-Хоквингема (БЧХ-коды) составляют один из больших классов кодов, исправляющих независимые ошибки веса t, метод построения которых может быть задан явно. Теоретически коды БЧХ могут исправлять произвольное количество ошибок, но при этом существенно увеличивается длительность кодовой комбинации, что приводит к уменьшению

скорости передачи данных и усложнению приемо-передающей аппаратуры (схем кодеров и декодеров) [19].

Методика построения кодов БЧХ отличается от обычных циклических, в основном, выбором определяющего полинома *P(x)*. Коды БЧХ строятся по заданной длине кодового слова *n* и числа исправляемых ошибок *S* , при этом количество информационных разрядов *k* не известно пока не выбран определяющий полином.

Коды БЧХ представляют собой циклические коды и, следовательно, к ним применимы любые методы декодирования циклических кодов. Открытие кодов БЧХ привело к необходимости поиска новых алгоритмов и методов реализации кодеров и декодеров. Получены существенно лучшие алгоритмы, специально разработанные для кодов БЧХ. Это алгоритмы Питерсона, Бэрлекэмпа и др.

Рассмотрим алгоритм Питерсона-Горенстейна-Цирлера [19]. Пусть БЧХ код над полем *GF(q)* длины *n* и с конструктивным расстоянием *d* задается порождающим полиномом *g(x)*, который имеет среди своих корней элементы $\beta^{l_0}, \beta^{l_0+1}, ..., \beta^{l_0+d-2}$ $\beta \in GF(q^m), \beta^n = 1,$ l_0 – целое число (например 0 или 1).

Тогда каждое кодовое слово обладает тем свойством, что $c(\beta^{l_0-1+j}) = 0, \quad j = 1,2,...,d-1.$

Принятое слово *r(x)* можно записать как *r(x)* = *c(x)* + *e(x)*, где *e(x)* – полином ошибок. Пусть произошло $u \le t = \dfrac{(d-1)}{2}$ ошибок на позициях $i_1, i_2, ..., i_u$ (*t* – максимальное число исправляемых ошибок), значит $e(x) = e_{i1}x^{i1} + e_{i2}x^{t2} + ... + e_{iw}x^{iw}$, а $e_{i1}, e_{i2}, ..., e_{iu}$ – величины ошибок.

Можно составить *j*-ый синдром S_j принятого слова *r(x)*:

$$S_j = r(\beta^{l_0-1+j}) = e(\beta^{l_0-1+j}), \quad j = 1,...,d-1 \qquad (10)$$

Задача состоит в нахождений числа ошибок *u*, их позиций $i_1, i_2, ..., i_u$ и их значений $e_{i1,} e_{i2}, ..., e_{iu}$ при известных синдромах S_j.

Предположим, для начала, что *u* в точности равно *t*. Запишем (10) в виде системы нелинейных уравнений в явном виде:

$$\begin{cases} S_1 = e_{i_1}\beta^{l_0 i_1} + e_{i_2}\beta^{l_0 i_2} + ... + e_{i_t}\beta^{l_0 i_t} \\ S_2 = e_{i_1}\beta^{(l_0+1) i_1} + e_{i_2}\beta^{(l_0+1) i_2} + ... + e_{i_t}\beta^{(l_0+1) i_t} \\ \qquad\qquad ..... \\ S_{d-1} = e_{i_1}\beta^{(l_0+d-2) i_1} + e_{i_2}\beta^{(l_0+d-2) i_2} + ... + e_{i_t}\beta^{(l_0+d-2) i_t} \end{cases} . \qquad (11)$$

Обозначим через $X_k = \beta^{i_k}$ локатор *k*-ой ошибки, а через $Y_k = e_{ik}$ – величину ошибки, $k = 1,...,t$. При этом все X_k различны, так как порядок элемента β равен *n*, и поэтому при известном X_k можно определить i_k как $i_k = \log_\beta X_k$:

$$\begin{cases} S_1 = Y_1 X_1^{t_0} + Y_2 X_2^{t_0} + ... + Y_t X_t^{t_0} \\ S_2 = Y_1 X_1^{t_0+1} + Y_2 X_2^{t_0+1} + ... + Y_t X_t^{t_0+1} \\ \qquad\qquad ..... \\ S_{d-1} = Y_1 X_1^{t_0+d-2} + Y_2 X_2^{t_0+d-2} + ... + Y_t X_t^{t_0+d-2} \end{cases} \qquad (12)$$

Составим полином локаторов ошибок:

$$\Lambda(x) = (1 - xX_1)(1 - xX_2)....(1 - xX_t) = \Lambda_t x^t + \Lambda_{t-1} x^{t-1} + ... + \Lambda_1 x + 1 . \qquad (13)$$

Корнями этого полинома являются элементы, обратные локаторам ошибок. Помножим обе части этого полинома на $Y_l X_l^{\vartheta+t}$. Полученное равенство будет справедливо для $\vartheta = l_0, l_0+1,..., l_0+d-1$, $l = 1,...,t$:

$$\Lambda(x)YX = \Lambda_t x^t Y_l X_l^{\vartheta+t} + \Lambda_{t-1} x^{t-1} Y_l X_l^{\vartheta+t} + ... + \Lambda_1 x Y_l X_l^{\vartheta+t} + Y_l X_l^{\vartheta+t} . \quad (14)$$

Положим $x = X_l^{-1}$ и подставим в (14). Получится равенство, справедливое для каждого $l \in 1,2,...,t$ и при всех $\vartheta \in l_0, l_0+1,..., l_0+d-1$:

$$0 = \Lambda_t Y_l X_l^{\vartheta} + \Lambda_{t-1} Y_l X_l^{\vartheta+1} + ... + \Lambda_1 Y_l X_l^{\vartheta+t-1} + Y_l X_l^{\vartheta+t} . \quad (15)$$

Таким образом, для каждого *l* можно записать свое равенство. Если их просуммировать по *l*, то получиться равенство, справедливое для каждого $\vartheta \in l_0, l_0+1,..., l_0+d-1$:

$$0 = \Lambda_t \sum_{l=1}^{t} Y_l X_l^{\vartheta} + \Lambda_{t-1} \sum_{l=1}^{t} Y_t X_t^{\vartheta+1} + ... + \Lambda_1 \sum_{l=1}^{t} Y_t X_t^{\vartheta+t-1} + \sum_{l=1}^{t} Y_t X_t^{\vartheta+t} . \quad (16)$$

Учитывая (13) и то, что $S_{j+p} = \sum_{l=1}^{t} Y_l X_l^{l_0+j+p-1} = \sum_{l=1}^{t} Y_l X_l^{\vartheta+p}$, $j = 1,...,d-1$, $\vartheta = l_0 + j - 1$, (то есть ϑ меняется в тех же пределах, что и ранее) получаем систему линейных уравнений:

$$\begin{cases} S_1\Lambda_t + S_2\Lambda_{t-1} + ... + S_t\Lambda_1 = -S_{t+1} \\ S_2\Lambda_t + S_3\Lambda_{t-1} + ... + S_{t+1}\Lambda_1 = -S_{t+2} \\ \qquad\qquad ..... \\ S_t\Lambda_t + S_{t+1}\Lambda_{t-1} + ... + S_{2t-1}\Lambda_1 = -S_{2t} \end{cases} . \quad (17)$$

Или в матричной форме:

$S^{(t)}\overline{\Lambda}^{(t)} = -\overline{S}^{(t)}$,

где

$$S^{(t)} = \begin{bmatrix} S_1 & S_2 & ... & S_t \\ S_2 & S_3 & ... & S_{t+1} \\ ... & ... & ... & \\ S_t & S_{t+1} & ... & S_{2t-2} \end{bmatrix}, \quad (18)$$

$$\overline{\Lambda}^{(t)} = \begin{bmatrix} \Lambda_t \\ \Lambda_{t-1} \\ ... \\ \Lambda_1 \end{bmatrix}, \quad \overline{S}^{(t)} = \begin{bmatrix} S_{t+1} \\ S_{t+2} \\ ... \\ S_{2t} \end{bmatrix}. \quad (19)$$

Если число ошибок и в самом деле равно *t*, то система (16) разрешима, и можно найти значения коэффициентов $\Lambda_1,...,\Lambda_t$. Если же число $u < t$, то определитель матрицы $S(t)$ системы (17) будет равен 0. Это есть признак того, что количество ошибок меньше *t*. Поэтому необходимо составить систему (16), предполагая число ошибок равным $t-1$. Высчитать определитель новой матрицы $S(t-1)$ и т.д., до тех пор, пока не установим истинное число ошибок.

После этого можно решить систему (17) и получить коэффициенты полинома локаторов ошибок. Его корни (элементы, обратные локаторам ошибок) можно найти простым перебором по всем элементам поля $GF(q^m)$ $GF(q^m)$. К ним найти элементы, обратные по умножению, – это локаторы

ошибок X_k, $k = 1,...u$. По локаторам можно найти позиции ошибок $(i_k = \log \beta X_k)$, а значения Y_k Y_kошибок из системы (11), приняв $t = u$. Декодирование завершено.

Множества решений системы линейных уравнений по *mod 2*

Поиск путей решения проблемы надёжности хранения информации вновь обращает внимание на простые и известные точки зрения. Суть их состояла в том, что для линейных кодов процесс кодирования состоит в отображении множества сообщений (длины m) в множество кодов сообщений (длины $n = m + s$). Формирование порождающей матрицы с требуемыми свойствами приводит к необходимости решения линейных систем уравнений больших размерностей. Именно проблема размерности стала препятствием для реализации численных методов на имеющихся вычислительных мощностях 80-90 годов прошлого века. Сегодня арсенал вычислительных мощностей за счёт производительности вычислительной техники, новых методов вычислений (параллельных, конвейерных, частичных, квантовых), а также GRID технологий позволяет обратить внимание на возможность указанной точки зрения. Следует отметить, что несмотря на резкое увеличение производительности вычислительной техники и новых методов вычисления, тем не менее, прямой способ поиска эффективных матриц (порождающей проверочной и декодирования), не позволяет решить эту проблему. Однако исследуя структуры самих множеств (сообщений и кодов сообщений), и структуры эффективных матриц для допустимых размерностей (с точки зрения вычислительных возможностей) можно найти инварианты матриц независимых от (n, m). Такой подход требует разработки эксперимента и соответствующего математического и программного обеспечения. В данном разделе выделен один аспект, связанный с особенностью решения систем линейных уравнений по *mod 2* для выбора эффективной порождающей матрицы. Следует отметить, что речь идёт не о нахождении какого-то одного нетривиального решения, а о нахождении всего множества нетривиальных решений системы уравнений. При этом, требования к алгоритму нахождения полного множества решений системы состоят в жёстком ограничении на время поиска и на объём необходимой памяти при поиске решений. Полное множество решений системы уравнений является основой для решения многих прикладных задач. Например, в задачах передачи данных по каналам, или в задачах хранения данных полное множество решений обеспечивает формирование с требуемыми свойствами механизмов кодирования, проверки, исправления и декодирования. Свойства множеств двоичных кортежей и структурные особенности системы уравнений позволяют резко сократить прямой перебор при поиске возможных решений. В статье предлагается метод частичных вычислений для формирования полного множества решений системы линейных уравнений по *mod 2*. При этом в методе используются параллельность и конвейерность вычислений.

Рассмотрим систему линейных уравнений:

$$\begin{cases} c_{11}x_1 + c_{12}x_2 + ... + c_{1j}x_j + ... + c_{1n}x_n = b_1 \\ c_{21}x_1 + c_{22}x_2 + ... + c_{2j}x_j + ... + c_{2n}x_n = b_2 \\ ..... \\ c_{j1}x_1 + c_{i2}x_2 + ... + c_{ij}x_j + ... + c_{in}x_n = b_i \\ ..... \\ c_{m1}x_1 + c_{m2}x_2 + ... + c_{mj}x_j + ... + c_{mn}x_n = b_m \end{cases}, \text{ где } c_i, b_i, x_i \in \{0,1\},\ n > m\ \ (n = m + s) \qquad (20)$$

Частные случаи системы (168) представляются как:

$$\begin{cases} c_{11}x_1 + c_{12}x_2 + ... + c_{1j}x_j + ... + c_{1n}x_n = 0 \\ c_{21}x_1 + c_{22}x_2 + ... + c_{2j}x_j + ... + c_{2n}x_n = 0 \\ ..... \\ c_{j1}x_1 + c_{i2}x_2 + ... + c_{ij}x_j + ... + c_{in}x_n = 0 \\ ..... \\ c_{m1}x_1 + c_{m2}x_2 + ... + c_{mj}x_j + ... + c_{mn}x_n = 0 \end{cases}, \text{ где } c_i, x_i \in \{0,1\},\ n > m\ \ (n = m + s). \tag{21}$$

$$\begin{cases} c_{11}x_1 + c_{12}x_2 + ... + c_{1j}x_j + ... + c_{1n}x_n = 1 \\ c_{21}x_1 + c_{22}x_2 + ... + c_{2j}x_j + ... + c_{2n}x_n = 1 \\ ..... \\ c_{j1}x_1 + c_{i2}x_2 + ... + c_{ij}x_j + ... + c_{in}x_n = 1 \\ ..... \\ c_{m1}x_1 + c_{m2}x_2 + ... + c_{mj}x_j + ... + c_{mn}x_n = 1 \end{cases}, \text{ где } c_i, x_i \in \{0,1\},\ n > m\ \ (n = m + s). \tag{22}$$

Суммирование переменных осуществляется по *mod 2*.

Рассмотрим алгоритм нахождения всех решений уравнения для частных случаев (21) и (22). Идея нахождения полного множества решений состоит в следующем. Пусть для первого уравнения системы известно множество (M1) решений. В этом случае остальные уравнения $(m-1)$ можно рассматривать как фильтры. Применим второе уравнение к множеству M_1 т.е. подставим все решения первого уравнения во второе. В результате получим множество M_{12}, причём возможно $M_{12} \subseteq M_1 \, и \, M_{12} \neq \varnothing$, либо $M_{12} = \varnothing$. Если $M_{12} = M_1$ – то первое и второе уравнения системы линейно зависимы. Если $M_{12} = \varnothing$ – то первое и второе уравнение системы несовместны. Если $M_{12} \subset M_1$ – это означает совместность уравнений, имеющих общее множество решений M_{12}. Для случая $M_{12} \subset M_1$ применим третье уравнение системы к множеству M_{12}. Получим множество M_{123}. При этом возможно: $M_{123} = M_{12}$ или $M_{123} = \varnothing$ или $M_{123} \subset M_{12}$. В случае $M_{12...i-1} \subset M_{12...i-2}$ применим i-ое уравнение к множеству $M_{123...i-1}$. Таким образом, последовательное применение уравнений системы к соответствующим множествам решений позволяет получить следующий результат: система линейных уравнений либо имеет единственное множество решений, либо не имеет решений. При единственности решения выполняется условие $M_{12...m} \subseteq M_{12...m-1} \subseteq ... \subseteq M_1$. Лишними операциями этой процедуры являются операции с уравнениями, которые являются линейно зависимыми. Если под показателем (ϑ) процедуры понимать две величины:

$$\tau = \sum_{i=1}^{m} \Delta_i, \text{ где } \tau - \text{общее время формирования } M_{12...m}, \tag{23}$$

Ошибка! Объект не может быть создан из кодов полей редактирования. (24)

где v – общий объём памяти необходимый для хранения множеств решений.

Так как $v_i \leq v_{i-1}$, можно считать, что $v = v_1$, таким образом $\vartheta = \vartheta(\tau, v)$.

Пусть k – число линейно зависимых уравнений в системе (20). Будем считать, что известно условие (u) проверки на линейную независимость уравнений системы. Время на проверку независимости уравнений равно Δ_u. Применяя условие независимости уравнений системы, удалим

из системы *k* зависимых уравнений. В этом случае общее время формирования множества решений системы определится как (при условии совместимости уравнений):

$$\tau = \Delta_u + \sum_{i=1}^{m-k} \Delta_i \ .$$

Если считать, что система линейных уравнений совместна и не содержит линейно зависимых уравнений, тогда время определения множества решений и требуемый объём памяти определится по соотношениям (23), (24). В этом случае показатель процедуры будет эффективным. Возможность параллельности вычислений для этой процедуры реализуется только для определения начального множества M_1.

Рассмотрим следующую процедуру формирования множества решений системы.

Сформируем множества $M'_1, M'_2, ..., M'_i, ...M'_m$, где M'_i – множество, полученное применением *i*-го уравнения к M_1. Для формирования этих множеств требуется:

$$\tau = \sum_{i=1}^{m} \Delta_{1i} = m\Delta, \text{ где } \Delta - \text{ время формирования } M'_i \ .$$

В этом случае имеет место конвейерность вычислений множеств M'_i.

Сформируем пары множеств (M'_{2k}, M'_{2k+1}), $k = 1, 2, ... \dfrac{m-1}{2}$. Найдём пересечение этих множеств. Для полученных множеств построим пары, для них найдем пересечения и т.д.

В результате получим множество решений системы уравнений. Такая процедура обеспечивает параллельность вычислений множеств решений. В этом случае при определённых условиях, время затрачиваемое на формирование множества решений будет меньше, чем для предыдущей процедуры. Но объём используемой памяти будет больше. Пусть время нахождения на *i*-ом уровне общего множества ($M_{ij} = M'_{ij} \cap M'_{ij+1}$) равно σ. Общее время формирования (λ) конечного множества решений определится как:

$$\lambda = \sum_{k=1}^{q} \frac{m-1}{2^k} \sigma .$$

Определим коэффициент эффективности этой процедуры с предыдущей как:

$$\theta = \frac{\lambda}{\tau} \le L \ (L - \text{заданная константа}) . \qquad (25)$$

Условие (25) позволяет сравнить эффективности рассмотренных процедур. Рассмотрим ещё одну процедуру. Пусть известно множество M_1 и число элементов равно *«p»*. Пусть имеется ровно «p» процессоров, на которых параллельно осуществляется проверка принадлежности элемента множества M_1 к множеству решения (M^*) системы (20). Время необходимое для проверки принадлежности элемента M_1 к M^* равно Δ. Тогда общее время формирования множества решений будет равно $t = \Delta$. Пусть число процессоров (k) равно $k = \dfrac{n}{2^s}$, $s = 0,1,...i,...,\ln(n)$. Тогда общее время формирования множества M^* равно $t = 2^s \Delta = 2^s$.

Пример:

n	delta	smax
1000	1	6

s	k	t
0	1000	1
1	500	2
2	250	4
3	125	8
4	62	16
5	31	32
6	15	64

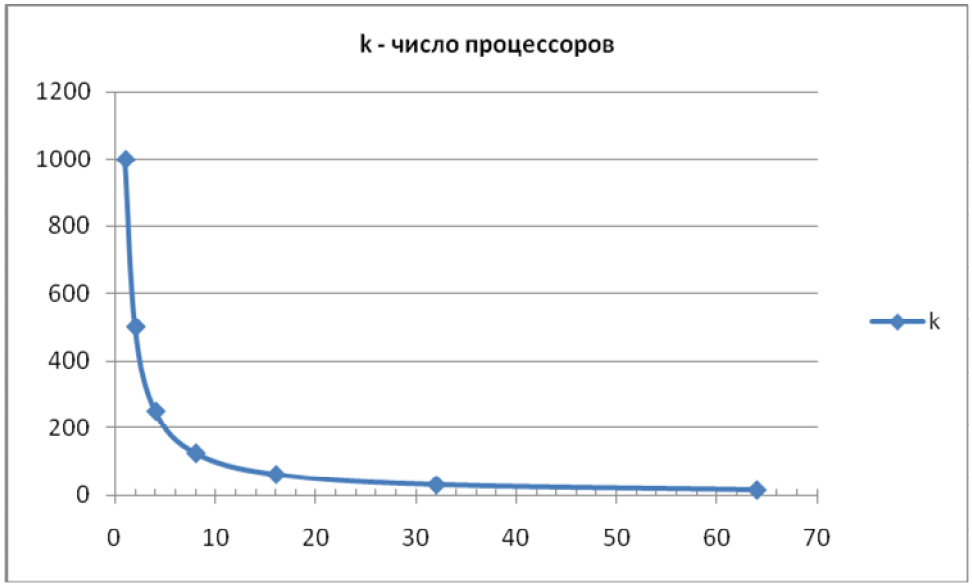


Рис. 9. Изменение числа процессоров

На рис. 10 и рис.11 представлены алгоритмы процедур.

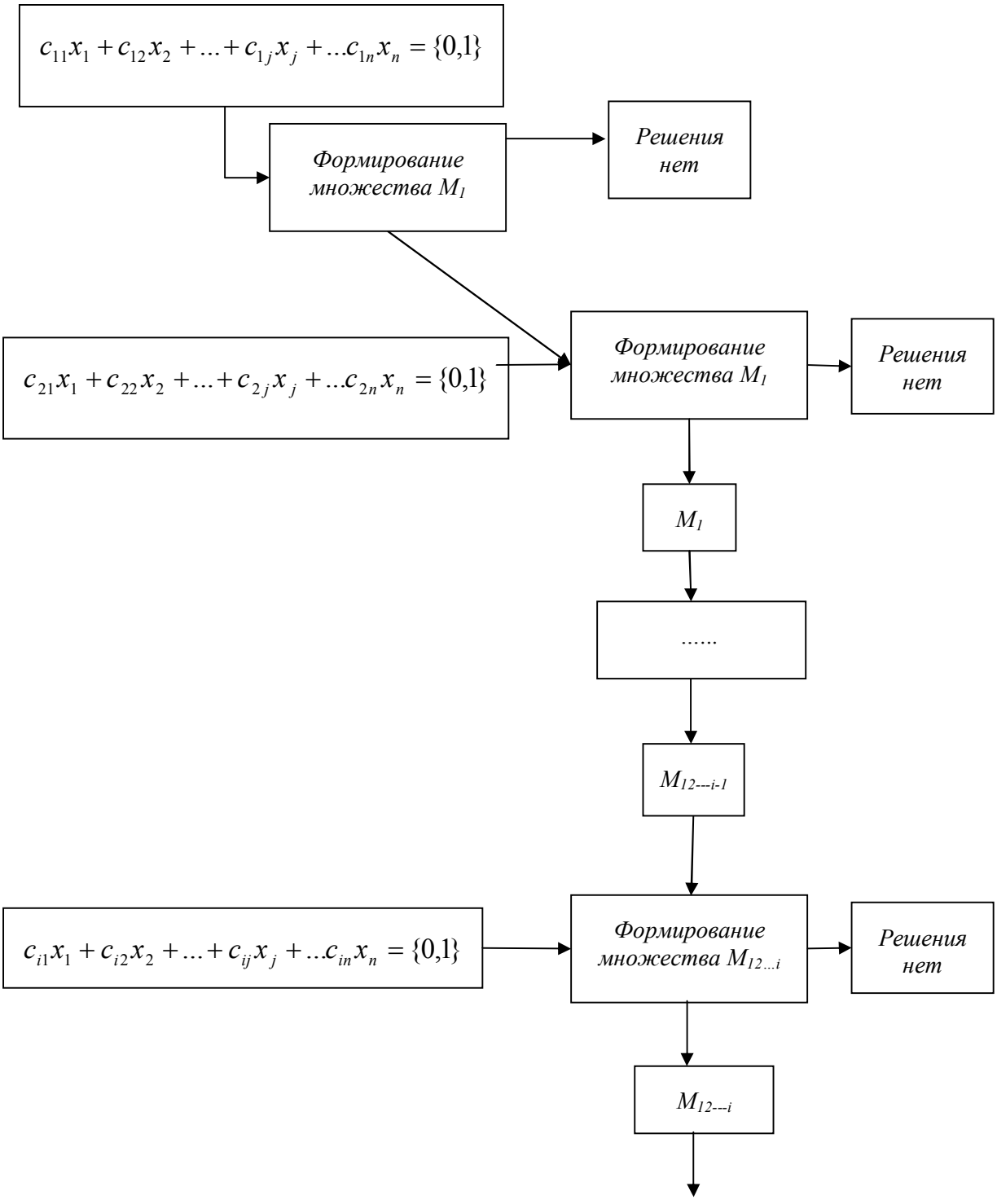


Рис. 10. Процедура 1 формирования множества решений системы

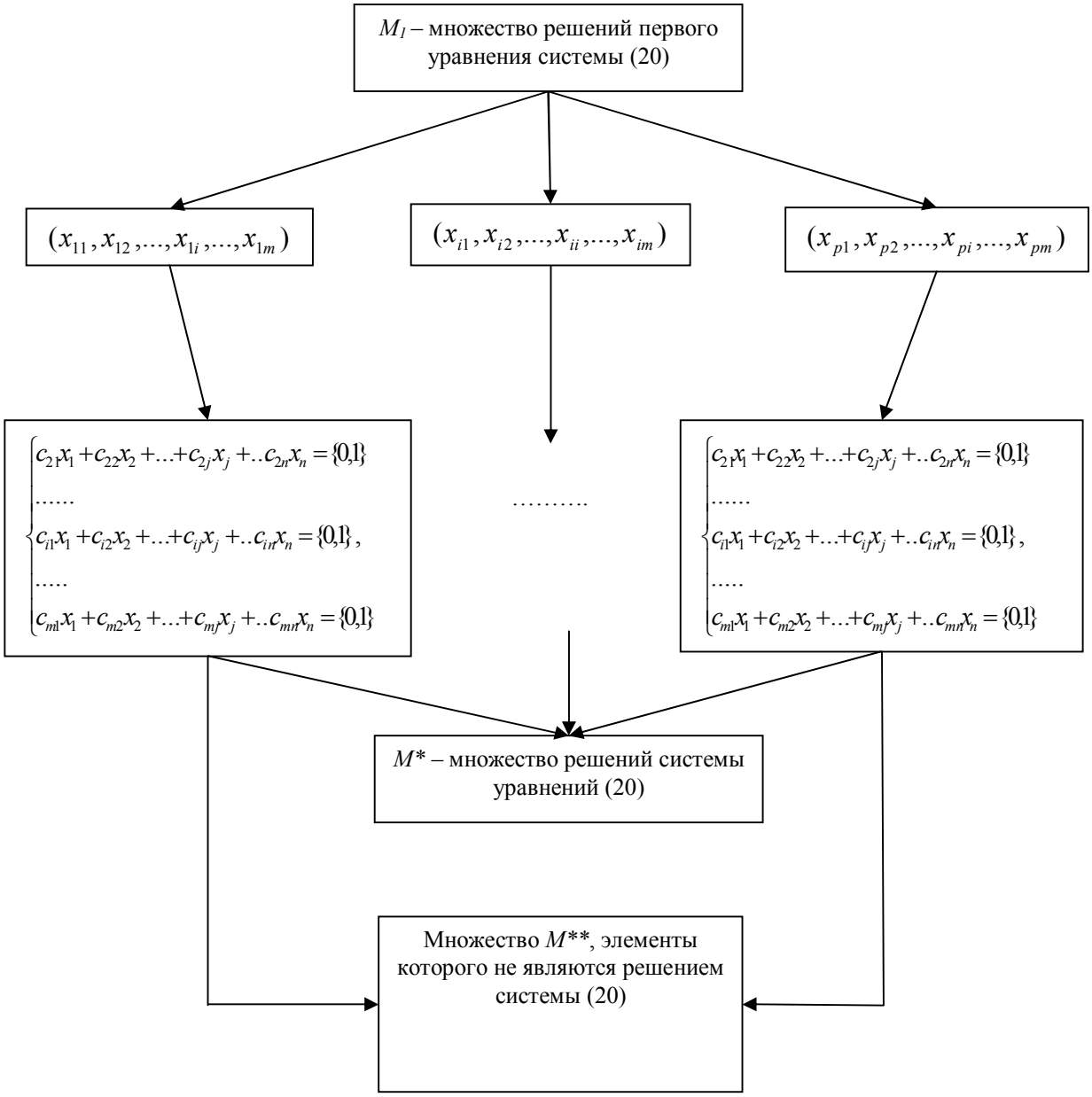


Рис. 11. Процедура 2 параллельно конвейерного формирования множества решений системы уравнений

Если имеется *«p»* (*p* – число элементов множества *M*) процессоров определения того, что элемент множества *M* является решением системы (20) и время определения равно Δ, тогда время формирования множества решений системы (если оно существует) равно $t = p\Delta$. Следует иметь в виду, что Δ – это общее время параллельного определения принадлежности элемента *M* общему *(M*)* решению системы (20). Если число процессоров меньше чем число элементов в M_1, тогда общее время построения множества *M** увеличивается.

Метод решения линейного уравнения по *mod 2*

Рассмотрим линейное уравнение:

$$c_1x_1 + c_2x_2 + ... + c_jx_j + ...c_nx_n = 0\,. \qquad (26)$$

Введём обозначения:

$C = (c_1, c_2, ..., c_i, ..., c_n)$ – список коэффициентов уравнения (26);

n_1 – число ненулевых коэффициентов уравнения (26);

n_2 – число нулевых коэффициентов уравнения (26) ($n = n_1 + n_2$);

$S_1 = (s_{11}, s_{12}, ...s_{1i}, ...s_{1n_1})$ – список индексов элементов списка C не равных нулю т.е. $C(S_1[s_{1i}]) = c_i = 1$;

$S_2 = (s_{21}, s_{22}, ...s_{2i}, ...s_{2n_2})$ – список индексов элементов списка C равных нулю, т.е. $C(S_2[s_{2j}]) = c_j = 0$.

Рассмотрим случаи:

$n_1 = n,\ n_2 = 0$. Обозначим через M_1 – множество всех решений уравнения (26);

$n_2 = n,\ n_1 = 0$. Обозначим через M_2 – множество всех решений уравнения (26);

$n_1, n_2 > 0$. Обозначим через M_3 – множество всех решений уравнения (26).

Для последнего случая рассмотрим варианты:

a) n_1 – чётное;

b) n_1 – нечётное.

Обозначим через:

$k_1 = \left[\frac{n_1}{2}\right] - 1,$ – число групп решений множества M_3 , когда n_1 – чётное;

$k_2 = \left[\frac{n_1}{2}\right],$ – число групп решений множества M_3 , когда n_1 – нечётное;

$$k = \begin{cases} k_1,\ n_1 \text{ чётное} \\ k_2,\ n_1 \text{ нечётное} \end{cases}.$$

С учётом свойства операции сложения по *mod 2* множество M_3 будет состоять из групп решений: $G_1, G_2, ..., G_i, ..., G_k$.

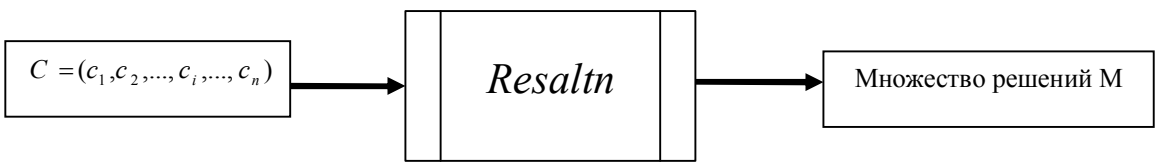
Первый элемент группы G_1 – $(0,0,...,0,...1,1)$, содержащий две единицы справа.

Последний элемент группы G_1 – $(1,1,...,0,...0,0)$, содержащий две единицы слева.

Первый элемент группы G_i – $(0,0,...,0,...,1,1,...,1,1)$, содержащий чётное *i* единиц справа.

Последний элемент группы G_i – $(1,1,...,1,...,0,0,...,0,0)$, содержащий чётное *i* единиц слева.

Введём обозначение: *Resaltn* – функция построения множества решений уравнения (26).



Рассмотрим уравнение $c_1x_1 + c_2x_2 + ... + c_jx_j + ...c_nx_n = 1$.

Сумма неизвестных переменных по *mod 2* должна быть равна единице, при соответствующих коэффициентах равных единице т.е.

$$\sum_{i=1}^{k} x_i = 1,\ (k - \textit{нечётное}),\ c_i = 1.$$

Полное множество решений состоит из следующих подмножеств кортежей:

1-ое подмножество содержит кортежи, состоящие из одной единицы и остальных нулей,

i-ое подмножество из *2i+1* единиц и остальных нулей.

Рассмотрим уравнение $c_1x_1 + c_2x_2 + \ldots + c_jx_j + \ldots c_nx_n = 0$.

Пусть некоторые переменные равны единице, при соответствующих коэффициентах равных тоже единице.

Возможны случаи:

a) число таких переменных чётное (*k*). В этом случае уравнение будет содержать меньшее число неизвестных (*n – k*)*;*

b) число переменных нечётное (*p*). В этом случае число неизвестных в уравнении будет равно *n* – (*p – 1*) и одна переменная будет равна единице. В этом случае приходим к уравнению вида:

$$c_1x_1 + c_2x_2 + 1 + \ldots + c_jx_j + \ldots c_nx_{n-(p-1)} = 0.$$

Рассмотрим уравнение $c_1x_1 + c_2x_2 + \ldots + c_jx_j + \ldots c_nx_n = 1$. Аналогично случаю с чётным и нечётным n_1.

Формирование множества решений уравнения (26) сводится к генерации множества двоичных кортежей, содержащее различные группы *G*. Опишем структуру генератора двоичных кортежей. Пусть:

GENn – имя функции генератора;

n – длина двоичного кортежа;

M – множество двоичных кортежей длины *n*;

Функция генерации состоит из следующих шагов.

Определяется число подмножеств двоичных кортежей *k = 1 + n.*

Для каждого подмножества M_i (подмножество содержит кортежи с *i*- единицами) определяется мощность подмножества S_i.

Считая, что M_i упорядочено по возрастанию и что первый элемент известен итерационно формируются последующие элементы. Поскольку последний элемент M_i известен тем самым завершение итерации происходит, когда последующий генерируемый элемент становится равным конечному.

Нахождение нетривиальных решений уравнения (26) осуществляется следующей функцией.

SOLn – имя функции формирования множества решений уравнения (26);

n – число неизвестных уравнения;

Cn – кортеж коэффициентов уравнения;

b – правая часть уравнения (b={0,1});

Xk – список позиций переменных, значения которых равны единице (список может отсутствовать);

M – множество решений уравнения;

m – размерность *M.*

Функция построения множества решений состоит из следующих шагов.

Определяются особенности уравнения (наличие переменных, имеющие единичные значения; нулевые и единичные коэффициенты; значение правоq части уравнения – нуль или единица)

Для каждой особенности формируется условие генерации множества решений (*M*).

С учётом условия генерации осуществляется генерация множества решений, с использованием функции *GENn*.

Как отмечалось множество *M* включает $n-1$ подмножеств с различным количеством единиц и два единичных подмножества с единственными кортежами (нулевой и единичный). Поэтому возможна реализация $n-1$ параллельных процессов формирования подмножеств множества *M*.

Заключение

В статье рассмотрены проблемы создания высоконадёжных электронных хранилищ информации, представлена история развития носителей информации, а также описана возможность применения корректирующих ошибки кодов, использующихся в сетях передачи информации для современных систем хранения. Дается описание наиболее эффективных существующих кодов, исправляющих ошибки, а также позволяющих их детектировать, описываются особенности использования каждого вида кода. Теоретические аспекты, связанные с методами надёжности хранения информации рассматриваются с применением теоретико-множественного подхода, что позволяет выделить множество изучаемых объектов и их основные свойства, эффективно реализовать задачу поиска кодирующих матриц, а методами решения систем линейных уравнений по модулю два построить алгоритм вычисления матриц проверки четности. Приводится описание процедур, необходимых для аппаратной реализации кодера и декодера использующих иррегулярные низкоплотные матрицы проверки четность. Решена задача распараллеливания этих процедур для повышения производительности реализующей их аппаратных систем. Что, в итоге, позволит проектировать эффективные и высоконадежные системы хранения информации.

Список литературы

1. Extracting Value from Chaos, June 2011. – [Электронный ресурс]. URL: http://idcdocserv.com/1142.
2. [Электронный ресурс]. – URL: http://www.emc.com/collateral/demos/microsites/emc-digital-universe-2011/index.htm.
3. [Электронный ресурс]. – URL: http://www.oracle.com/us/products/servers-storage/storage/tape-storage/sl8500-modular-library-system/overview/index.html.
4. Солтанов А.Г. Схемы декодирования и оценка эффективности LDPC-кодов. Применение, преимущества и перспективы развития// Безопасность информационных технологий. – М.: 2010. – №2. – С. 61-68.
5. [Электронный ресурс]. – URL: http://recoverme.ru/index.php?option=com_content&viewarticle&id=90%3A2011-12-27-14-27-02&catid=6%3Apolezno&Itemid=3.
6. [Электронный ресурс]. – URL: http://www.acronis.ru/glossary/backup/cloning.
7. [Электронный ресурс]. – URL: http://www.acronis.ru/glossary/backup/drive-image.
8. Романов Д. Sun Microsystems: управляемая сеть хранения данных //«Storage News», июль 2000.
9. [Электронный ресурс]. – URL: http://virtlibe.narod.ru/Psyhologyupr/psih.pdf.
10. Mirko Lorenz. Vision Cloud: The Fast Sheet. – [Электронный ресурс]. – URL: http://www.visioncloud.eu/content.php?s=30,47.
11. Гипервизоры систем хранения данных //Журнал сетевых решений/LAN. – 2012. – №4.
12. [Электронный ресурс]. – URL: www.habrahabr.ru/post/111274/.

13. Кунегин С.В. Системы передачи информации // Курс лекций. – М. в/ч 33965, 1997. – С. 317.

14. Воробьев К. А. Методы построения и декодирования недвоичных низкоплотностных кодов // Теория и практика системного анализа. – 2010. – Т. II. –С. 96-102.

15. LDPC Codes, Application to Next Generation Communication Systems - Dr. Lin-Nan Lee Vice President. – Hughes Network Systems, Germantown, Maryland 20854. – October 8, 2003.

16. Гласман К. Цифровая видеотехника: кодирование, обнаруживающее и исправляющее ошибки // журнал «625», 2006.

17. Золотарев В. В., Овечкин Г. В. Обзор исследований и разработок методов помехоустойчивого кодирования (по состоянию на 2005 год).

18. [Электронный ресурс]. – URL: http://www.mtdbest.ru/articles/obzor_po_kodir2.pdf.

19. Архипкин А. Турбокоды – мощные алгоритмы для современных систем связи // Беспроводные технологии. – 2006. – №1.

20. Воробьев П.Е. Поиск эффективных стратегий кэширования данных». – М.: ФГУГНИИ ИТТ «Информика».

21. Галлагер Р. Теория Информации и Надежная Связь. – М.: Советское радио, 1974. – С. 720.

22. Витерби А.Д, Омура Дж.К. Принципы цифровой связи и кодирования. – М.: Радио и связь, 1982.