САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Быкова Юлия Сергеевна

Разработка модуля вычисления синдромов и восстановления утраченных дисков в RAID-массиве с использованием арифметики поля $\mathrm{GF}(2^{256})$

Курсовая работа

Научный руководитель:
Руководитель исследовательской лаборатории RAIDIX Платонов С. М.

Санкт-Петербург
2013

Оглавление

Введение

Объем информации, создаваемый человечеством, постоянно растет. По мнению ученых, если записать абсолютно все цифровые данные мира на компакт-диски и сложить эти диски, то получившаяся бы стопка простиралась в высоту на расстояние, равное расстоянию между Землей и Луной. И с каждым годом только увеличивается. Это объясняет спрос на системы хранения данных (СХД).

Система Хранения Данных (СХД) - это комплексное решение по организации надёжного хранения больших объемов информации, а так же быстрого и бесперебойного доступа к ней.

Существуют множество различных технологий СХД, и RAID(Redundant Array of Independent Disks) одна из них. RAID увеличивает скорость чтения и записи, распараллеливая их, и обеспечивает нам сохранность наших данных. В частности методика RAID 6 использует две различные контрольные суммы, что позволяет восстанавливать данные при выходе из строя до двух жестких дисков в дисковой группе.

Как известно, время - деньги. Нам критична скорость получения информации. Обращение к диску не достаточно быстрая операция. Если правильно организовать вычисления, то нам быстрее будет восстановить данные на «запаздывающих» дисках, чем прочитать их. Этот факт увеличивает пользу от наших исследований.

1. Основные термины и технологии

У нас есть дисковый массив. Каждый диск мы разбиваем на блоки одинакового размера, так, чтобы размер блока являлся делителем размера диска. Блоки с одинаковыми порядковыми номерами формируют страйп. Таким образом в каждом страйпе содержатся ровно по одному блоку из каждого диска. В каждом страйпе выделяется по два блока на контрольные суммы, которые мы будем называть синдромами. Не умоляя общности можно считать, что синдромы расположены в последних двух блоках страйпа. Уровнем абстракции расчетного модуля является один страйп.

В наших исследованиях мы используем арифметику полей Галуа. С точки зрения практического подхода, поле Галуа порядка 2^n является полем, элементы которого мы можем представлять как вектора длины n, состоящие из 0 и 1. В то же время элементы можно представить в виде многочлена степени n с коэффициентами из поля вычетов по модулю 2 ($mod 2$). С последним представление мы и будем работать. В этом случае нам полезно определить примитивный элемент поля – элемент, степени которого пробегают по всем ненулевым элементам поля. В нашем представлении в качестве него можно взять многочлен x. Так мы и поступим.

2. Обоснование выбора подхода и поля

Существует множество циклических кодов, предназначенных для преобразования информации и защиты ее от ошибок. Коды Рида-Соломона являются одними из самых распространенных. И этому есть свое объяснение.

Используя этот подход мы можем свести все операции по подсчету синдромов и восстановления данных к простым операциям сложения и умножения на примитивный элемент. Также этот способ кодирования позволяет выявлять скрытые повреждения и восстанавливать данные после них.

Выбор поля Галуа $GF(2^{256})$ так же не случаен и имеет ряд преимуществ:

- Большой размер элемента поля позволяет работать с огромным количеством дисков, около $1.16 * 10^{77}$, что примерно равно $0,12\%$ от количества всех атомов в обозримой Вселенной. Данное замечание полезно не только в RAID, но и в других областях, где требуется помехоустойчивое кодирование.[4]

- Возможность обработки страйпа за одну итерацию.

  Длина страйпа - 4Кб из-за технических особенностей, пришедших к нам с более высокого уровня абстракции. Представляя страйп как 256 последовательно расположенных переменных, мы получаем, что каждая переменная имеет размер 16 байт, что численно равняется размеру регистра при использовании технологии SSE.

- Эффективность операции умножения на x по сравнению с непараллельной реализацией.

  Благодаря эффективной реализации вычислений и использованию технологий SSE и AVX количество инструкций при одном умножении зависит только от количества единиц в двоичном представлении образующего многочлена поля. В нашем случае, на умножение на x большого числа элементов расходуется всего три процессорных инструкции.

3. Цели и задачи

Цель Проведение исследования возможности возникновения потенциального прироста производительности при переходе с полей Галуа размена 2^8 на поля размера 2^{256} по средством параллельной обработки большого числа элементов за очень малое число векторных операций.

Задачи С помощью кода Рида-Соломона эффективно реализовать вычисление синдромов с элементами в поле $GF(2^{256})$.

Сравнить производительность по сравнению с полями размера 2^8 и 2^{16}.

4. Алгоритм расчёта синдромов

При работе с кодами Рида-Соломона используются следующие формулы [1]:

$$P = D_0 + D_1 + D_2 + \cdots + D_n$$

$$Q = D_0 x^b + D_1 x^{n-1} + D_2 x^{n-2} + \cdots + D_n$$

Где:

- D_i – блок с данными i-того диска RAID-массива;
- P, Q – значения вычисленных функций от блоков данных всех дисков, они и будут формировать первый и второй синдромы;
- x – примитивный элемент поля $GF(2^{256})$.

Используя схему Хорнера, можно представить формулу вычисления Q так, чтобы в ней использовались только операции умножения на примитивный элемент поля и сложения.

$$Q = D_0 + x * (D_1 + x * (D_2 + ... + x * (D_{n-2} + x * D_{n-1})...))$$

Умножение на произвольный элемент поля также сводится через факторизацию умножения на к сложениям в зависимости от битов элемента, на который умножаешь

$a(x)*b(x) = (a_{n-1}*x^{n-1}+a_{n-2}*x^{n-2}+...+a_1*x+a_0)*(b_(n-1)*x^(n-1)+ b_{n-2}*x^{n-2}+ ...+b_1*x+b_0) = b_0*a(x)+x*(b_1*a(x+...+x*(b_{n-3}*a(x)+x*(b_{n-2}*a(x)+x*b_{n-1}*a(x)))...)$

где b_i – это биты элемента, на который производится умножение.

Стоит заметить, что в качестве сложения в полях Галуа используется операция побитового исключающего ИЛИ (xor) [5].

При реализации умножения нам важен выбор неприводимого многочлена в поле. Чем меньше единиц в коэффициентах нашего неприводимого многочлена, тем меньше операций необходимо для реализации умножения на . Одним их многочленов, удовлетворяющих этому требованию является $f(x) = x^{256} + x^{16} + x^3 + x + 1$.

Умножение произвольного элемента на приметивный элемент x выглядит следующим образом:

$$f(x) * a_{n-1} + \tilde{a}$$

Где $\tilde{a}$ - логически сдвинутый на еденицу влево элемент a

5. Реализация

Известно, что условные переходы и циклы замедляют работу программы. Поэтому мы поставили себе задачу: линеаризовать код модуля. Для достижения этого для каждого числа дисков от 5 до 128 был реализован свой набор функций вычисления и перерасчета синдромов и восстановления данных. Функции, служащие одним целям, были объединены в соответствующие массивы. Предполагается, что получится много однотипного кода. Разумно автоматизировать процесс написания кода, используя код-генератор.

Тем не менее, до конца линеаризовать код не получится из-за большого размера исполняемых файлов. Часть циклов пришлось оставить, на порядок уменьшив объем файлов. К примеру размер кода функции вычисления синдромов уменьшился с 240 Мб до 1.09 Мб.

Для написания генератора был выбран язык C, по следующим причинам:

1. Простота переносимости кода.

   В случае перехода на другую архитектуру выполняющего процессора, нам будет достаточно переписать определяющие функции (сложение, умножение на примитивный элемент поля).

2. Возможности использования оптимизаций компилятора.

   Современные компиляторы способны грамотно преобразовать код, распределить память и уменьшить размер модуля.

3. Упрощенная работа с регистрами.

   Компилятор берет работу по распределению данных на регистры на себя, освобождая нас от этого. [2] [3]

4. Приятным дополнением является простота реализации и тестирования на языке C относительно ассемблера.

   Для всех возможных конфигураций систем (от 5 до 128) производился замер времени работы алгоритма в тиках процессора. Для этого была применена функция `RDTSC()`. Тест проводился 2000 раз. 5% лучших и худших результатов отбрасывалось. Для сравнения на графиках использовалось выборочное среднее.

   Используемая конфигурация тестовой машины:

   - ОС: Debian 7.0
   - Тип ОС: x64
   - CPU: Intel® Xeon(R) CPU E5-2620 0 @ 2.00GHz × 18
   - Оперативная память: 39.4 Гб

Код собирался компилятором gcc 4.7 с опцией оптимизации –O3.

6. Расчет синдромов

Ниже приведен график сравнений нашей реализации с аналогами:

MatrixN2 – алгоритм, основанный на идее взаимного соответствия элементов поля Галуа и некоторого множества матриц. Реализован на ассемблере.

ARF64 – алгоритм, использующий инструкции процессоров x86-64, предназначенные для работы с полями Галуа. Основан на работе с полиномами 64-ой степени.

GF8,GF16 – реализации алгоритма, аналогичного нашему, но использующие поля размера 2^8 и 2^{16} соответственно.

На всех графиках по оси X отложено количество дисков, а по оси Y – время одного вызова функции в тысячах тиков процессора.

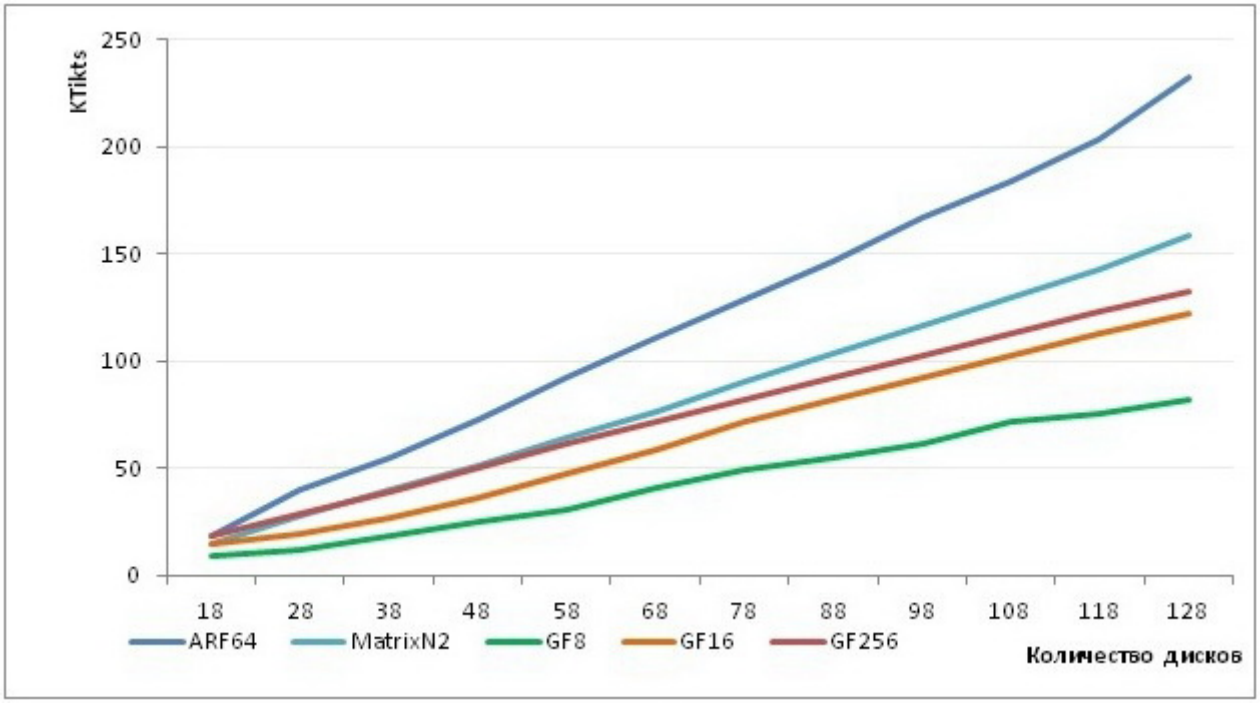

Диаграмма 1: Расчет двух синдромов

Можно увидеть, что наш алгоритм выигрывает по скорости у MatrixN2 и ARF64, но работает хуже своих аналогов.

Предположительно это связано с нехваткой регистров и недостаточным размером кэша процессора.

Операции над полем $GF(2^{256})$ также производятся существенно медленнее, чем в полях размера 2^8 и 2^{16}.

7. Результаты

В данной работе рассматривался подход к вычислениям в RAID с использованием арифметики конечных полей $GF(2^{256})$. Полученные результаты ниже ожидаемых. Исследовалось влияние технических ограничений на производительность алгоритма. Разработаны код-генераторы для генерации кода функций расчета двух синдромов. Подготовлена среда испытаний, программа тестов производительности. Проведены тесты корректности и замеры производительности получившихся функций. Выполнен анализ полученных результатов. Сделано и проверено предположение о недостаточной производительности алгоритмов, выявлены некоторые причины этого.

8. Планы на дальнейшую работу

В дальнейшем, планируется написать функции для расчета трех синдромов и, соответственно, для восстановления трех дисков, а также функции для восстановления скрытых повреждений информации.

Но данное напрвление не является приоритетным, поскольку у него нет перспектив развития в ближайшее время.

В будущем, возможно, появятся процессоры с большим объёмом кэша, на которых данный алгоритм проявит себя значительно лучше.

Также планируется рассмотреть работу функций в полях $GF(2^{64})$ и $GF(2^{128})$.

Список литературы

[1] Anvin H. Peter. The mathematics of RAID-6. — 2011.

[2] The Software Optimization Cookbook High-Performance Recipes for IA-32 Platform / Richard Gerber, Aart J.C. Bik, Kevin B. Smith, Xinmin Tian. Second Edition. — Intel Press, 2006.

[3] Кнут Д.Э. Искусство программирования. Т.2: Полученные алгоритмы. — Издательство Вильямс, 2004.

[4] Питерсон У., Уэлдон Э. Коды, исправляющие ошибки. — М. : Мир, 1976.

[5] Утешев А.Ю. Поля Галуа. — http://pmpu.ru/vf4/gruppe/galois.