САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Савельев Николай Геннадьевич

Разработка модуля вычисления синдромов в RAID-массиве с использованием арифметики поля $GF(2^{16})$

Курсовая работа

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

Санкт-Петербург
2013

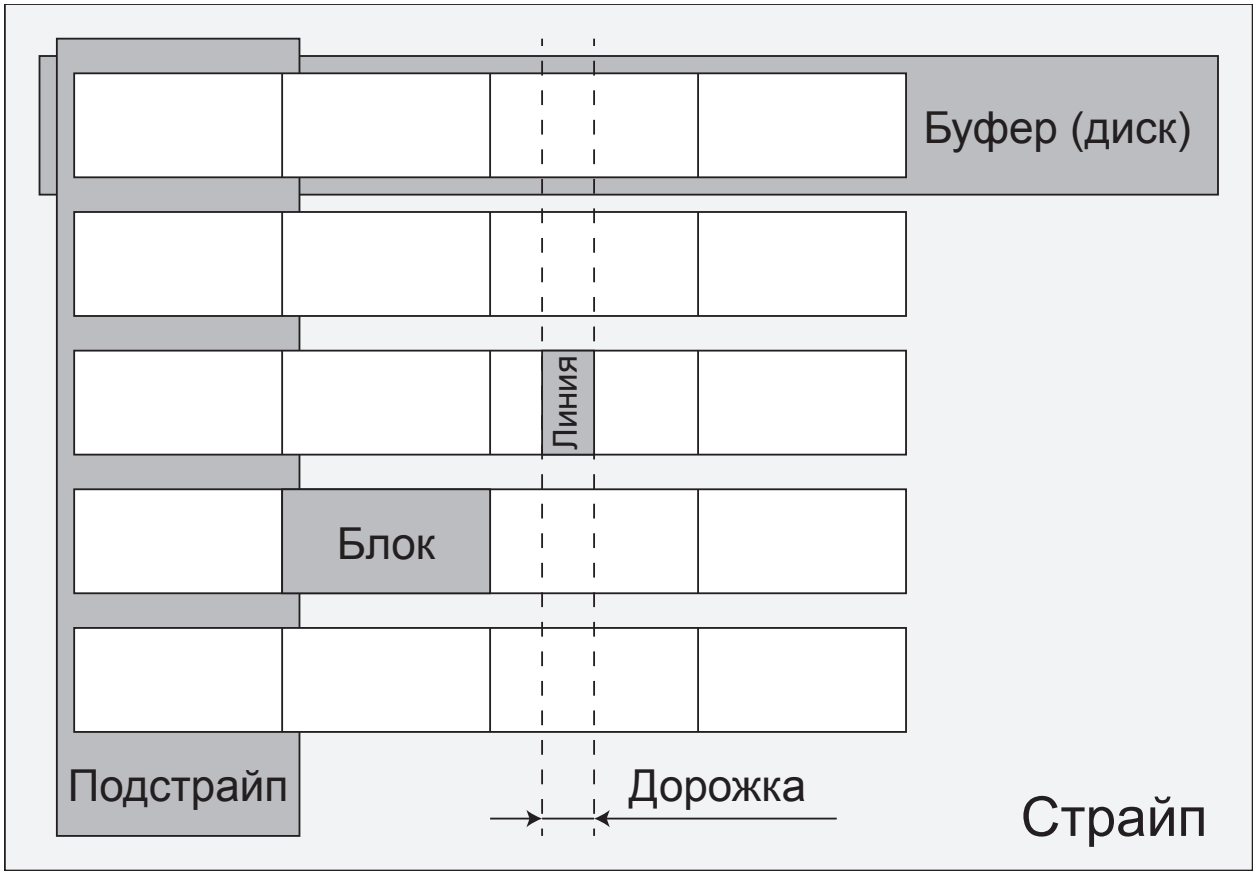
Оглавление

Введение

Объём хранимой информации растёт быстрыми темпами [2]. В течение последних пяти лет он показал десятикратный прирост. Для хранения огромных объемов информации, которые уже исчисляются зеттабайтами, используются системы хранения данных на основе технологии RAID. Эта технология позволяет значительно увеличить объем хранимой информации, распределяет нагрузку на носители информации, обеспечивает параллельный доступ к хранилищам для чтения и записи и отказоустойчивость, которая основана на применении контрольных сумм.

В последнее время появляются все более производительные накопители данных, например SSD (Solid State Drive), которые выводят скорость доступа к данных на более высокий уровень. В связи с этим возникает необходимость в увеличении быстродействия алгоритмов работы с RAID-массивами. На текущий момент наиболее распространены алгоритмы RAID с использованием кодов Рида-Соломона в полях Галуа размера 2^8. Целью моей работы являлось исследовать возможный прирост производительности при переходе от поля Галуа размера 2^8 к полю размера 2^{16}, а также применении векторных инструкций процессора для параллельных вычислений.

1. Терминология



Страйп – основная единица обработки данных в системе хранения.

Буфер – одна из частей одинакового размера, на которые разбит страйп. Обозначается $D_0, D_1, D_2, \ldots, D_{N-1}$, где N — количество буферов — равно количеству дисков данных в массиве. В рамках данной работы буфера могут называться **дисками**.

Синдромы – дополнительные буфера (диски), предназначенные для обеспечения отказоустойчивости. Размер синдромов равен размеру дисков данных. Значения синдромов используются в случае потери данных для их восстановления.

Блоки – части буферов одинакового размера, такое разбиение необходимо для организации вычислений. Размер блока является делителем размера буфера. В СХД компании RAIDIX, как правило, используется размер буфера, равный четырём килобайтам.

Подстрайп – совокупность блоков разных буферов с одинаковыми последовательными номерами. Каждой функции нижнего уровня, работающей непосредственно с данными и синдромами, передаётся один подстрайп.

Линии – части блоков одинакового размера. Размер линии является делителем размера блока и зависит от мощности используемого поля. В нашем случае он составлял 256 байт.

Дорожка – объединение линий с одинаковыми номерами в подстрайпе из разных блоков. Является единицей данных, обрабатываемой функциями расчёта за один вызов.

2. Актуальность задачи

Реализованный алгоритм расчета синдромов имеет следующие преимущества:

Увеличение лимита на количество дисков в RAID-массиве по сравнению с полем $GF(2^8)$. Больший размер элемента поля позволяет работать с 65535 дисками вместо 255. Так как задача расчёта с использованием кодов Рида-Соломона важна не только для RAID, то снятие этого ограничения даёт возможность использовать разработанный алгоритм в других областях, где требуется помехоустойчивое кодирование, таких, как системы цифровой связи[4].

Эффективность операции умножения на примитивный элемент поля по сравнению с непараллельной реализацией. Благодаря эффективной реализации вычислений и использованию технологий SSE и AVX количество инструкций, которые нужны для одного умножения, зависит только от количества единиц в двоичном представлении примитивного элемента поля. В нашем случае, для того, чтобы умножить на x 128 или 256 элементов, процессору потребуется выполнить всего три векторные инструкции.

Увеличение количества информации, обрабатываемой за один вызов функции расчета по сравнению с полем $GF(2^8)$. Из-за удвоения размеров элементов поля возможно за один проход обрабатывать вдвое больший объём данных.

3. Алгоритм расчёта синдромов

Введём обозначения:

- 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.

Расчет двух синдромов происходит по формулам[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 x – примитивный элемент поля $GF(2^{16})$, в качестве которого мы взяли `0x100B`.

Так как вычисление этих формул является узким местом в плане производительности в нашей работе, то одной из наших задач являлась оптимизация данного процесса.

В качестве сложения в полях Галуа используется операция побитового исключающего ИЛИ[5].

Для второй формулы, воспользовавшись факторизацией, можно вывести более производительное решение:

$$Q = x \cdot (\cdots x \cdot (x \cdot (D_0) + D_1) + D_2) + \cdots) + D_n$$

Таким образом вычисление многочлена от блоков с данными сводится к двум операциям: сложению и умножению на примитивный элемент поля в терминах этого поля. Прирост производительности ожидался за счёт повышения эффективности умножения на x, а также обработки большего участка памяти за один вызов функции расчёта.

4. Используемые инструменты

Разработка велась на языке программирования C. Если бы мы использовали язык ассемблера, было бы неясно, как использовать регистры SSE/AVX. Например, для расчёта двух синдромов в нашей схеме векторных вычислений требуется 32 регистра, при том, что процессор предоставляет всего 16. Мы считаем, что компилятор C использует регистры более эффективно в отношении скорости работы, чем распределение их вручную[3]. К тому же, порядок использования регистров может быть самым разнообразным, что принесло бы трудности в случае реализации на языке ассемблера. Также плюсом C является переносимость кода на другие платформы, в частности, на ARM.

Был использован генератор кода. В связи с тем, что наличие в коде условных переходов негативно влияет на скорость его выполнения, было принято решение использовать отдельную функцию для каждого количества дисков в массиве (от 5 до 128) с последующим объединением их в массив. Генератор позволяет автоматизировать написание большого количества похожих функций. При этом значительно возрастает объём исполняемого файла, но также увеличивается и скорость выполнения алгоритмов.

5. Тестирование

Состав процедуры тестирования:

1. Для выбранных количества дисков D и размеров страйпа S выделяется область памяти размера $(S \cdot D)$, которая впоследствие заполняется случайными данными. Потом она логически разделяется на D одинаковых блоков, эмулирующих различные жесткие диски.

2. Запускается функция расчёта контрольных сумм, которые сохраняются в служебных блоках.

3. Проверяется работа функций восстановления сбойных блоков, реализованная моим коллегой по курсовой работе Ильей Демьяненко:

   3.1. Случайным образом выбираются номера блоков, чье повреждение будет эмулироваться.

   3.2. Данные из этих блоков сохраняются в отдельном массиве.

   3.3. Эти блоки заполняются случайными данными.

   3.4. Вызывается функция восстановления дисков, которая использует вычисленные контрольные суммы.

   3.5. Проверяется совпадение данных, сохранённых в массиве, и восстановленных функцией.

Замеры времени проводились следующим образом: до и после вызова тестируемой функции выполняется команда `RDTSC()`, возвращающая значение счётчика тиков процессора; использование этой команды позволило добиться высокой точности измерений. Для каждой функции тест проводился 2000 раз, затем обрасывались по 5% самых худших и лучших результатов. Из оставшихся берётся среднее значение.

Характеристики тестового сервера:

OC: Debian 7.0 x64

CPU: Intel Xeon®E5-2620 @ 2.00GHZ × 18

RAM: 39.4GiB

6. Результаты измерений

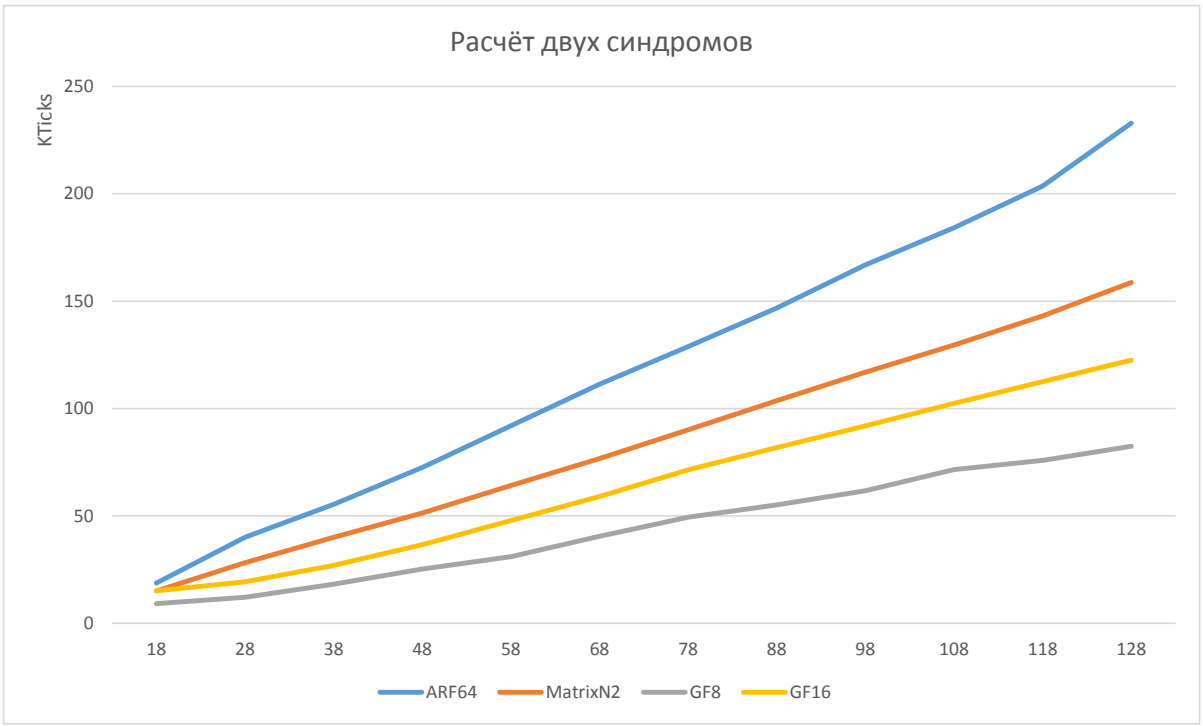
Ниже приведены графики сравнений нашей реализации с аналогами:

ARF64 – алгоритм, реализованный на языке ассемблера и использующий инструкции процессоров x86-64, предназначенные для работы с полями Галуа.

MatrixN2 – алгоритм, реализованный на языке ассемблера и основанный на идее взаимно однозначного соответствия между элементами полей Галуа и некоторого множества матриц.

GF8 – реализация алгоритма, аналогичного нашему, но использующая поле размера 2^8.

На графике по оси X обозначено количество дисков, по оси Y – время одного вызова функции расчета синдромов в тысячах тиков процессора.



Как мы видим, наша реализация выигрывает в скорости у предыдущих, но проигрывает аналогу для меньшего поля. По моему мнению, это связано с нехваткой регистров, так как за один проход функцией расчета синдромов обрабатывается до 32 переменных, и недостаточным размером кэша процессора, а следовательно, с большим количеством обращений к памяти.

Заключение

В данной работе рассмотрена реализация модуля расчета синдромов для RAID-массивов на основе векторных вычислений в поле $GF(2^{16})$. Полученные результаты ниже ожидаемых. При этом не исключено, что в будущем появятся процессоры с большим объёмом кэша, на которых данный алгоритм проявит себя значительно лучше. Возможные направления дальнейших исследований – оценка производительности других функций (работа с тремя синдромами, выявление и устранение скрытых повреждений диска), а также эксперименты с $GF(2^p)$ при простых p, что позволит снизить количество операций при умножении на x до одной.

Список литературы

[1] Anvin H. P. The mathematics of RAID-6. — 2006-2011. — URL: http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf.

[2] G. John F., C. Christopher, M. Alex et al. The Diverse and Exploding Digital Universe. — URL: http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf.

[3] The Software Optimization Cookbook: High Performance Recipes for IA-32 Platforms, 2nd Edition / R. Gerber, K. Smith, A. J. C. Bik, X. Tian. — 2005.

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

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