САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Демьяненко Илья Игоревич,
Савельев Николай Геннадьевич

Разработка модуля вычисления синдромов и восстановления утраченных дисков 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

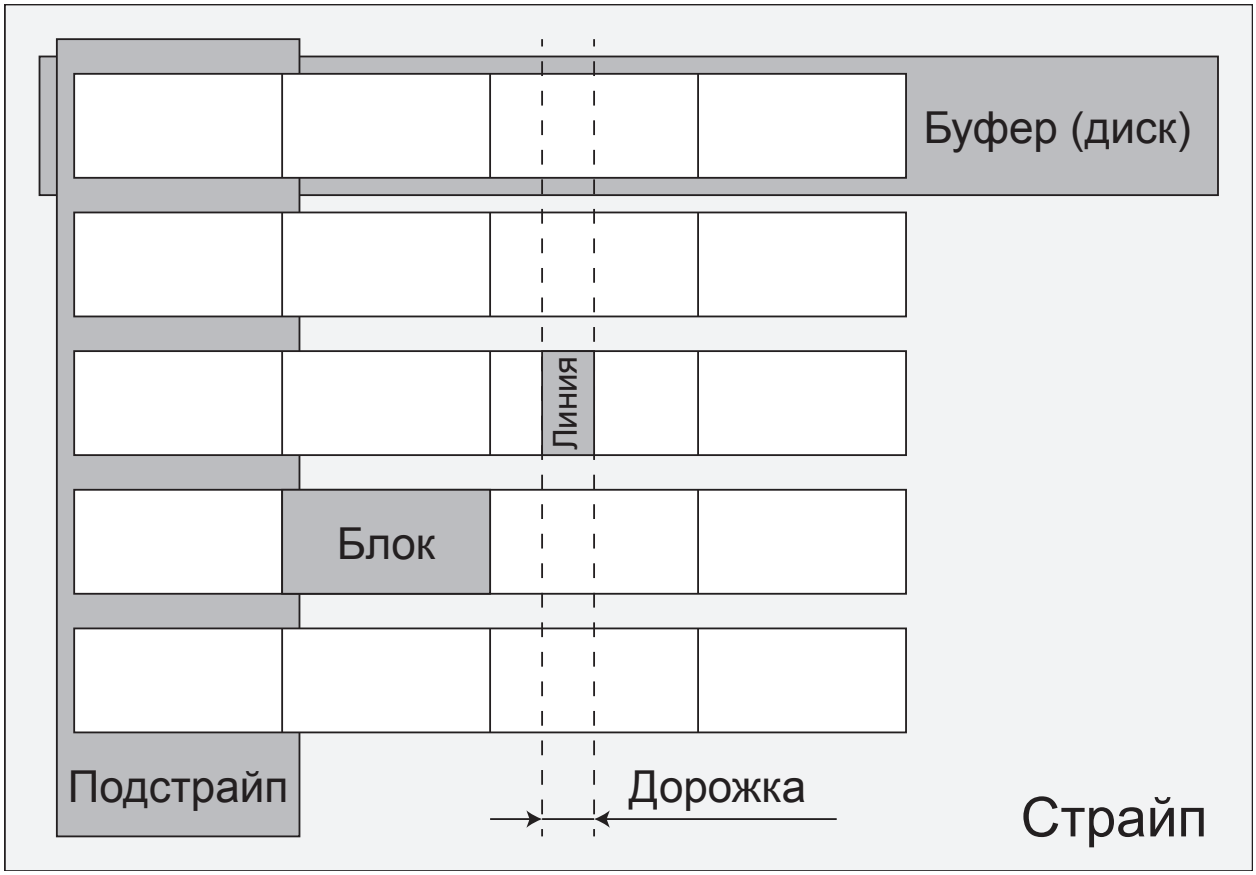
Введение

Как известно, объём информации, производимый человечеством, растёт экспоненциально [2]. В частности, за последние пять лет он увеличился в 10 раз и в данный момент измеряется зеттабайтами (ZB). Для хранения столь большого количества данных используются системы хранения данных (СХД).

Поскольку увеличиваются требования к объёму и производительности СХД, в них применяется технология RAID. Она увеличивает возможный размер хранилища, а также позволяет распараллелить процесс чтения-записи, что обеспечивает высокую скорость доступа к данным, а также предоставляет механизм отказоустойчивости благодаря контрольным суммам.

Из-за появления новых, более производительных накопителей данных, таких, как SSD (Solid State Drive), возникает необходимость в увеличении быстродействия алгоритмов работы с RAID-массивами. В данный момент наиболее распространённой реализацией алгоритмов RAID является таковая с использованием кодов Рида-Соломона в полях Галуа размера 2^8. Цель нашей работы состояла в исследовании возможного прироста производительности при переходе с полей Галуа размера 2^8 на поля размера 2^{16}, а также применении параллельных вычислений с использованием векторных инструкций процессора.

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 / восстановления за один вызов.

2. Актуальность задачи

Реализованные алгоритмы для работы с RAID-массивами имеют следующие преимущества:

Увеличение максимального количества дисков по сравнению с полем $GF(2^8)$. Больший размер элемента поля позволяет работать с 65535 дисками вместо 255. И поскольку задача расчёта и использования кодов Рида-Соломона важна не только для RAID, снятие этого ограничения даёт возможность использовать разработанные алгоритмы в других областях, где требуется помехоустойчивое кодирование, таких, как системы цифровой связи[4].

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 элемента поля. В нашем случае, на умножение на x 128 или 256 элементов расходуется всего три процессорных инструкции.

Увеличение объёма данных, обрабатываемых за один вызов функции по сравнению с полем $GF(2^8)$. Из-за удвоения размеров элементов поля за раз обрабатывается вдвое больший объём данных, используя для умножения на x то же число операций. Но существуют технические ограничения, такие как количество регистров и размер кэша процессора, которые не позволяют предсказать результат заранее.

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[6].

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. Алгоритм восстановления двух дисков

Формулы, используемые для восстановления двух дисков с данными[5]:

$D_j = \left(\left(Q + \bar{Q}\right) \cdot x^{-(n-k-1)} + P + \bar{P}\right)(x^{k-j} + 1)^{-1}$

$D_k = P + \bar{P} + D_j$, где j и k – номера вышедших из строя дисков, а $\bar{P}$ и $\bar{Q}$ – синдромы, рассчитанные без их учёта (соответствующие блоки заполнены нулями).

Для восстановления были рассчитаны таблицы степеней x и $(x^{k-j} + 1)^{-1}$ для всевозможных $k - j$, а также написана функция умножения набора элементов на произвольный элемент поля.

При каждом вызове функции восстановления страйпа ей передаются необходимые значения из таблицы, что позволяет поместить их в кэш при множественном вызове.

5. 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.

6. Тестирование

Процедура тестирования состояла из следующих этапов:

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

2. Запускается функция расчёта контрольных сумм, которых сохраняются в служебных блоках (это последние блоки в дисковом массиве[1]).

3. Проверяется работа функций восстановления сбойных блоков:

   3.1. Случайным образом выбираются номера блоков, чей выход из строя будет эмулироваться.

   3.2. Данные из этих блоков сохраняются в отдельном массиве.

   3.3. Эти блоки инициализируются случайными данными.

   3.4. Вызывается нужная функция восстановления дисков.

   3.5. Проверяется совпадение данных из восстановленных блоков и данных, сохранённых в массиве.

Замеры проводились следующим образом: до и после каждого вызова тестируемой функции вызывается команда `RDTSC()`, возвращающая значение счётчика процессорных тиков; использование этой команды позволило добиться высокой точности измерений. Для каждой функции тест проводился 2000 раз, затем обрасывались по 5% самых маленьких и самых больших 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

[1] В реальных системах они распределены по дискам равномерно, но мы можем считать их последними без потери корректности

7. Результаты измерений

Ниже приведены графики сравнений нашей реализации с аналогами:

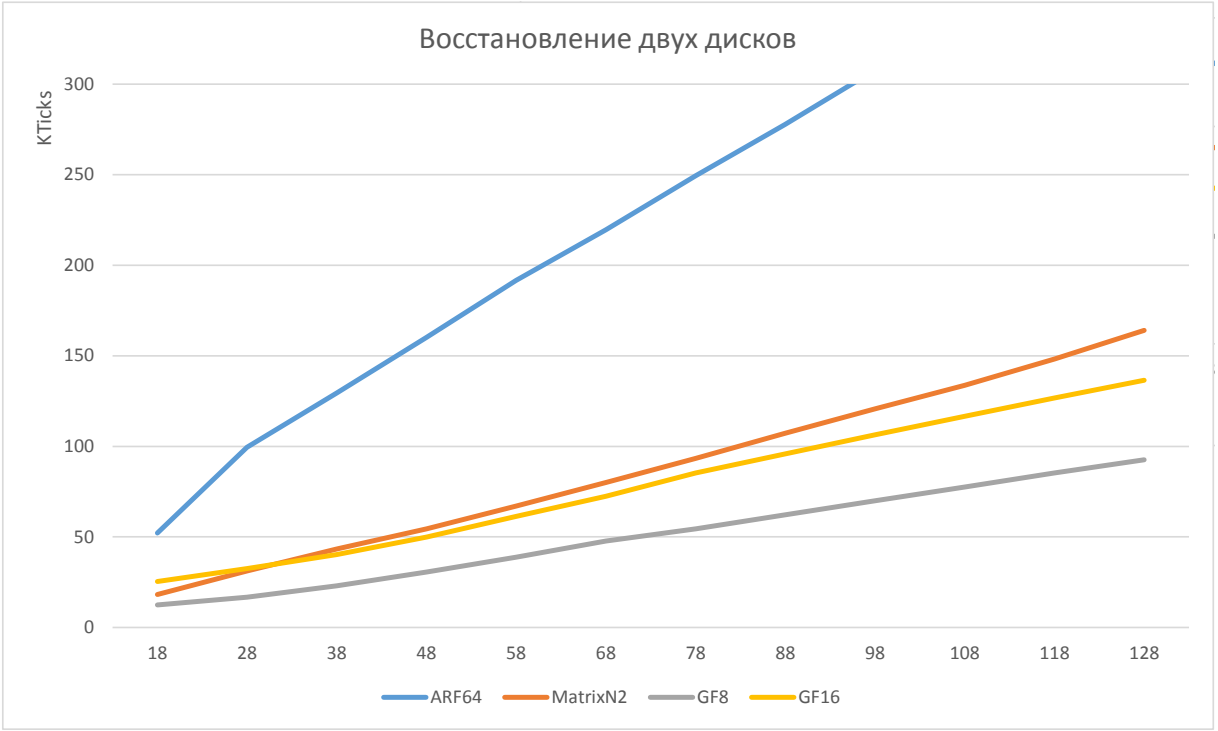
ARF64 – алгоритм, использующий инструкции процессоров x86-64, предназначенные для работы с полями Галуа. Реализован на языке ассемблера.

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

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

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

Как мы видим, наша реализация выигрывает в скорости у предыдущих, но проигрывает аналогу для меньшего поля.



При восстановлении дисков с данными наша реализация по-прежнему выигрывает у предыдущих, порой даже сильно, но также проигрывает реализации с $GF(2^8)$. По нашему мнению, это связано с сильной нехваткой регистров и недостаточным размером кэша процессора, а следовательно, с большим количеством обращений к памяти.

Заключение

В данной работе рассмотрена реализация модуля работы с RAID-массивами 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. Математика отказоустойчивых дисковых массивов. — URL: http://pmpu.ru/vf4/codes/raid.

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