САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Демьяненко Илья Игоревич, 344 группа

Разработка модуля 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

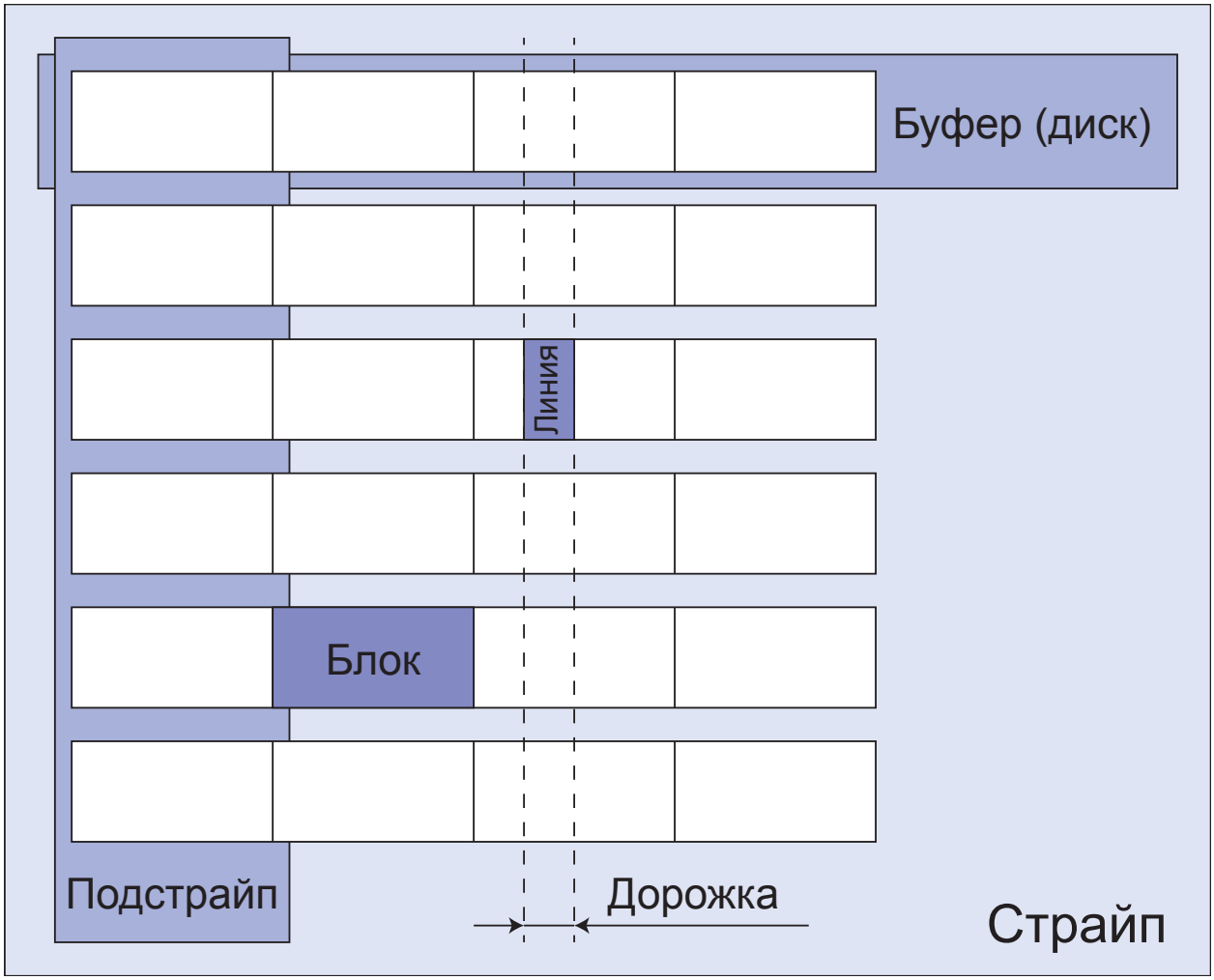
Введение

Как известно, объём информации, производимый человечеством, растёт экспоненциально [1]. 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 – множество буферов, входящих в состав дискового массива.

Буфер – одна из составляющих страйпа. В данной работе обозначены как $D_0, D_1, D_2, \ldots, D_{N-1}$, где N равно количеству дисков в массиве. Буфер по сути эквивалентен **диску**.

Синдромы – буфера (диски), хранящие контрольные суммы. Размер синдрома совпадает с объёмом диска данных. Используются для восстановления информации в случае её повреждения или выхода из строя диска с данными.

Блоки – составные части буферов, предназначенные для повышения удобства работы с буферами и распараллеливания вычислений. Их размер, как правило, равен четырём килобайтам.

Подстрайп – множество блоков с одинаковыми номерами, но но расположенными на разных дисках. Расчётные функции работают именно с ними.

Линии – составные части блоков. Являются минимальным объёмом данных с одного блока, для которого производится расчёт.

Дорожка – совокупность линий с одинаковыми номерами в подстрайпе из разных блоков. Обрабатываются расчётными функциями нижнего уровня 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[3].

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 одновременно 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 восстановления двух дисков

Формулы, используемые для восстановления двух дисков с данными[4]:

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

$$D_k = P + \bar{P} + D_j$$

В этих формулах j и k обозначают номера вышедших из строя дисков, а $\bar{P}$ и $\bar{Q}$ – синдромы, рассчитанные без их учёта (соответствующие блоки заполнены нулями).

Из формул следует, что для восстановления двух дисков необходимо вычислить две новых контрольных суммы, а также найти обратные элементы к $x^{-(n-k-1)}$ и $x^{k-j}+1$. Для избежания повторных вычислений были предподсчитаны таблицы таких элементов, а также написана функция умножения множества 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.

4. Используемые инструменты

Исходный код модуля восстановления дисков был написан на языке C. Несмотря на то, что высоконагруженные участки низкоуровневого кода обычно пишутся на языке ассемблера, что использование языка C имеет свои преимущества. Во-первых, он избавляет нас от ручного управления регистрами процессора. Для хранения контрольных сумм в идеале нужно 32 регистра SSE/AVX, при этом архитектура Intel®64, на которой проводились тесты, предоставляет только 16 таких регистров. Алгоритмы оптимизаций, используемые в современных компиляторах, учитывают внутреннее строение конкретного процессора и могут оптимальнее распределить синдромы по регистрам с учётом этой информации [2]. Во-вторых, при использовании языка C код может быть портирован на другие платформы, например, ARM, с минимальными изменениями.

Был применён кодогенератор. Для каждого возможного количества дисков в массиве (от 5 до 128) по задумке должна быть своя функция восстановления. Это позволяет реализовать их наиболее оптимальным образом для каждого конкретного случая, избегая использования условных переходов и бесполезных участков кода. Для того, чтобы не писать однотипные функции вручную, я реализовал их генератор, что также позволило вносить изменения, направленные на оптимизацию, во все функции сразу.

5. 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:

ОС: 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

6. Результаты измерений

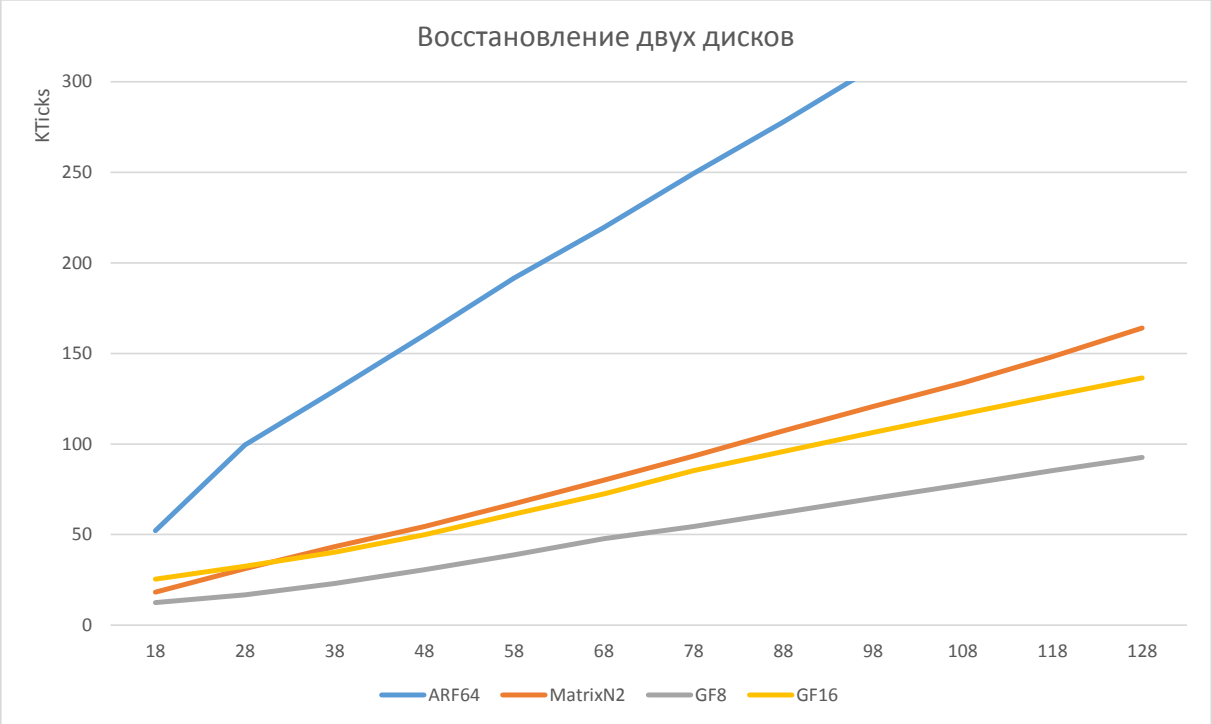
Ниже приведены графики сравнений нашей реализации с аналогами:

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 графике 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. 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 восстановления утраченных дисков в RAID-массивах на основе векторных вычислений в поле $GF(2^{16})$. Реализация на основе исследуемого поля проявила себя не так хорошо, как ожидалось. Причиной этому является сильная нехватка регистров, которая не компенсируется уменьшением числа операций при расчёте. Преодолеть её можно, используя модуль на RISC-архитектурах с б‘ольшим количеством регистров. В этом случае есть явные предпосылки к выигрышу у алгоритма, использующего поле меньшего размера.

Возможные направления дальнейших исследований:

- Реализация и тестирование производительности функции восстановления трёх дисков по трём синдромам, а также выявления и устранения скрытых повреждений диска. В них реализации с $GF(2^8)$ тоже перестанет хватать регистров, поэтому возможен выигрыш $GF(2^16)$.

- Применение полей $GF(2^p)$, где p – простые числа. Такие поля примечательны тем, что их образующий многочлен содержит минимальное количество единичных коэффициентов, поэтому умножение на x сразу 128 или 256 элементов поля будет производиться всего за одну инструкцию процессора.

Список литературы

[1] 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.

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

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

[4] Ю. Утешев А. Математика отказоустойчивых дисковых массивов. — URL: http://pmpu.ru/vf4/codes/raid.