Санкт-Петербургский государственный университет

Кафедра системного программирования
Математическое обеспечение и администрирование
информационных систем

Жереб Вера Вадимовна

Реализация алгоритмов RAID-вычислений для процессора ARM

Курсовая работа

Научный руководитель:
д. ф.-м. н. Терехов А. Н.

Консультант:
Руководитель Исследовательской Лаборатории RAIDIX
Маров А.В.

Санкт-Петербург
2020

Оглавление

Введение

Объёмы информации, которые хранит человек, растут с каждым годом. Для безопасного хранения данных и предоставления гарантированного доступа к ним применяется технология RAID (Redundant Array of Independent Disks) - массив из нескольких дисков, управляемый контроллером, «отказоустойчивый массив из независимых дисков».

Для обеспечения отказоустойчивости массивов на них записываются и хранятся не только данные, но и некоторая избыточная информация, которая позволяет восстанавливать данные в случае их частичной утраты.

Существует множество модификаций RAID (представлены лишь некоторые из них):

• RAID 0: данные делятся на столько частей, сколько дисков в массиве, и равномерно распределяются по дискам. Отказоустойчивость не предусмотрена.

• RAID 1 (зеркалирование): одни и те же данные записываются на два диска одновременно. Такой вариант хранения данных надёжен, но влечёт большую избыточность - 50%.

• RAID 5: в отличие от предыдущих способов хранения данных, RAID 5 допускает параллельную запись, поскольку блоки данных и контрольные суммы циклически записываются на все диски массива. Также RAID 5 позволяет восстановить до одного утраченного диска, если его номер заранее известен.

• RAID 6: данный способ, как и предыдущий, позволяет параллельную запись, однако избыточность в данной модификации на один диск больше, чем в RAID 5, так как возможно вос-

становление одного или двух утраченных дисков, а еще можно разрешить одно SDC (Silent Data Corruption) - повреждение, при котором нужно сначала найти место, в котором произошел сбой.

Так так технология RAID 6 дает больше возможностей для восстановления данных, в данной работе рассматривалась именно эта модификация.

В технологии RAID 6 выделяется несколько подзадач: расчет синдромов и восстановление утраченных дисков. Восстановление дисков может предусматривать как и то, что номера поврежденных дисков нам заранее известны, так и SDC. В рамках данной курсовой работы будет рассматриваться первый тип повреждений.

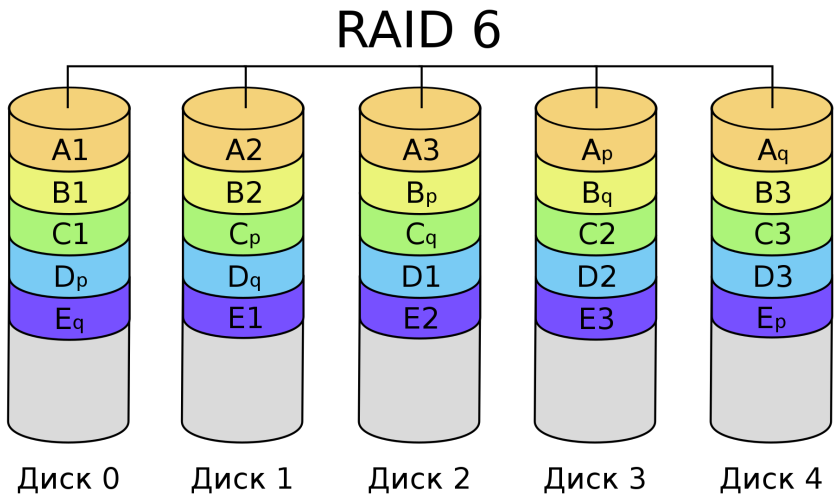

Рис. 1: Технология RAID 6. Источник: [6]

RAID-массивы могут создаваться аппаратно – с помощью RAID-контроллера и нескольких дисков, из которых и будет форми-

роваться массив. Также RAID-массивы могут формироваться программно – программа имитирует работу контроллера. При программной реализации затрагиваются ресурсы процессора и оперативной памяти, что снижает производительность ПК. Но в программной реализации скорость работы напрямую зависит от производительности процессора и нагрузки на него (соответственно, программная реализация RAID может выигрывать по скорости у аппаратной и обеспечивать ряд дополнительных полезных функций).

Рассмотрим ситуацию:

Если пользователю необходимо обеспечить взаимодействие данных из RAID-массива с другими устройствами в сети, эти данные поступают к необходимым устройствам через сетевую плату (NIC), например, с помощью сетевого коммутатора (Switch). Проблема заключается в том, что в представленной выше реализации велика нагрузка на центральный процессор.

В связи с потребностью уменьшить количество задач, выполняемых центральным процессором, были разработаны SmartNIC – сетевые интерфейсные карты (сетевые адаптеры), которые принимают на себя задачи обработки, обычно выполняемые центральным процессором. Используя собственный встроенный ARM-процессор, SmartNIC могут выполнять любую комбинацию шифрования/дешифрования, межсетевого экрана, TCP/IP и обработки HTTP [4].

Данная работа позволит уменьшить нагрузку на центральный процессор при помощи вынесения части логики технологии RAID в SmartNIC.

1. Цели и задачи

1.1. Цель работы

Реализация части технологии RAID (вычисление синдромов и восстановление поврежденных данных) на архитектуре ARMv8.

1.2. Поставленные задачи

Для выполнения обозначенной цели необходимо выполнить следующие задачи:

- Изучить предметную область.
- Реализовать три алгоритма RAID вычислений для Intel и для ARM: без векторизации, с векторизацией, алгоритм компании RAIDIX.
- Перенести алгоритмы RAID-вычислений в SmartNIC.
- Сравнить алгоритмы.

2. Обзор

2.1. Терминология

Каждый диск разделяется на блоки, размер которых будет удобен для дальнейшей обработки. Такие блоки называются ***стрипами***.
Стрипы, предназначенные для хранения данных, обозначаются $D_0, \ldots, D_{N-1}$, где N - количество дисков (два диска с синдромами не учитываются).
Синдромы - дополнительные стрипы, использующиеся для восстановления данных. Обозначаются P, Q.
Страйп - основная единица обработки данных в системе хранения. Каждый диск разделяется на блоки одинакового размера (стрипы), они нумеруются внутри одного диска, затем блоки с одинаковыми номерами из разных дисков формируют страйп.

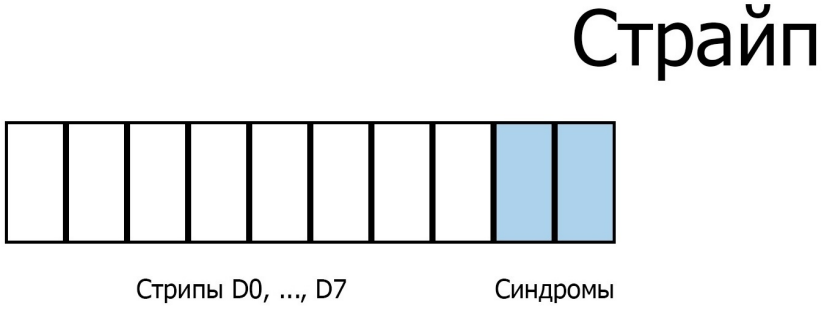

Рис. 2: Представление данных в страйпе, N = 8

2.2. Алгоритм расчёта синдромов и восстановления утраченных дисков

В основе алгоритма подсчета синдромов и восстановления синдромов лежит арифметика полей Галуа.

Для кодирования было выбрано поле Галуа $GF(2^8)$. Оно позволяет обрабатывать до 256 дисков, а для современных нужд такого количества достаточно, притом кодирование при увеличении числа дисков значительно усложняется.

Синдромы P и Q рассчитываются по следующим формулам (умножение на x и сложение производится по двойному модулю (2, неприводимый многочлен 8 степени):

$$P = \sum_{i=0}^{N-1} D_i$$

$$Q = \sum_{i=0}^{N-1} x^{N-i-1} D_i = (((D_0 x + D_1)x + D_2)x + \ldots + D_{N-1})$$

Утраченные блоки страйпа D_α и D_β, такие, что $\alpha<\beta$, восстанавливаются по следующим формулам:

$$D_\beta = \frac{\overline{P_{\alpha,\beta}} - \overline{Q_{\alpha,\beta}} x^{\alpha-N+1}}{1 - x^{\alpha-\beta}}$$

$$D_\alpha = \overline{P_{\alpha,\beta}} - D_\beta$$

где $\overline{P_{\alpha,\beta}}$ и $\overline{Q_{\alpha,\beta}}$ - пересчитанные P и Q для страйпа с утраченными дисками.

Заметим, что скорость чтения и записи данных напрямую зависит от скорости выполнения арифметических операций в полях

Галуа. В данной работе будет оптимизироваться выполнение этих операций.

2.3. Обзор существующих решений

Существует алгоритм, описанный James S. Plank, Kevin M. Greenan и Ethan L. Miller, который также активно применяется в разработке, например, компанией Intel. Подробное описание алгоритма представлено в [2]. Но этот алгоритм использует инструкцию shuffle для SSE. В Neon, к сожалению, не существует альтернативной инструкции. Возможна реализция без Shuffle, но это задействует большее количетво ресурсов и уменьшает производительность, поэтому в данной работе этот алгоритм не рассматривается.

Также стоит заметить, что две компании - Broadcom [1] и Mellanox [3] уже переносили часть RAID в свои SmartNIC, но это частные программные продукты, и получить код или детали реализации невозможно. Более того, SmartNIC от Broadcom и Mellanox только аппаратно поддерживают часть технологии RAID. Как следствие, они реализуют функции RAID (например, вычисление синдромов) с помощью библиотек или примитивов, которые работают в пользовательском пространстве. Представленный в данной работе алгоритм, наоборот, нацелен на реализацию в пространстве ядра. Таким образом, если мы будем пользоваться уже встроенными Broadcom или Mellanox библиотеками или примитивами, будет замедляться работа программы вследствие обмена между пользовательским пространством и пространством ядра.

3. Инструменты

3.1. Язык

Для разработки использовался язык программирования C. Этот язык был выбран по несколькими причинами:

• Язык программирования C является основным инструментом для написания функционала в ядре Linux и модулей ядра.

• Легкость написания кода и отладки, лучшая портируемость кода между различными архитектурами по сравнению с Ассемблером.

• Наличие Intrinsics - это функции, которые компилятор заменяет соответствующей инструкцией или последовательностью инструкций.

3.2. Векторные вычисления

Для разработки использовались команды, поддерживающие расширения SSE (для Intel), а также команды, использующие расширения Neon, так как разработка велась под архитектуру ARMv8.

SSE - это векторное расширение для архитектуры Intel, включающее в себя 16 (для 64-битных систем) 128-битных регистров XMM0-XMM15 и набор инструкций для них.

Neon - это векторное расширение для архитектуры ARM, комбинированный 64- и 128-битный набор команд SIMD (single instruction multiple data) - принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных. Neon обладает внушительным набором команд, отдельными регистро-

выми файлами, и независимой системой исполнения на аппаратном уровне.

4. Реализация

4.1. Реализованные алгоритмы

Реализованы алгоритмы, обозначенные в задаче:

Алгоритм без векторизации

В данном алгоритме мы рассматриваем отдельно каждый байт как один элемент поля $GF(2^8)$.

Плюсы: возможно применение на устройствах, не поддерживающих векторизацию.

Минусы: самый медленный из представленных алгоритмов.

Для всех блоков данных выполняются одинаковые действия. Это позволяет применять различные алгоритмы векторизации вычислений.

Алгоритм с векторизацией

Данный алгоритм рассматривает часть одного блока страйпа как шестнадцать последовательных многочленов из $GF(2^8)$.

Плюсы: быстрее, чем алгоритм без векторизации.

Минусы: требует поддержку векторизации.

Алгоритм компании RAIDIX

В отличие от предыдущих алгоритмов, в которых страйп разбивается на части, каждая из которых содержит элементы поля $GF(2^8)$, в данном алгоритме регистры мысленно располагаются "вертикально", и происходит обработка сразу 128 элементов поля.

Подробное описание алгоритма представлено в [5].

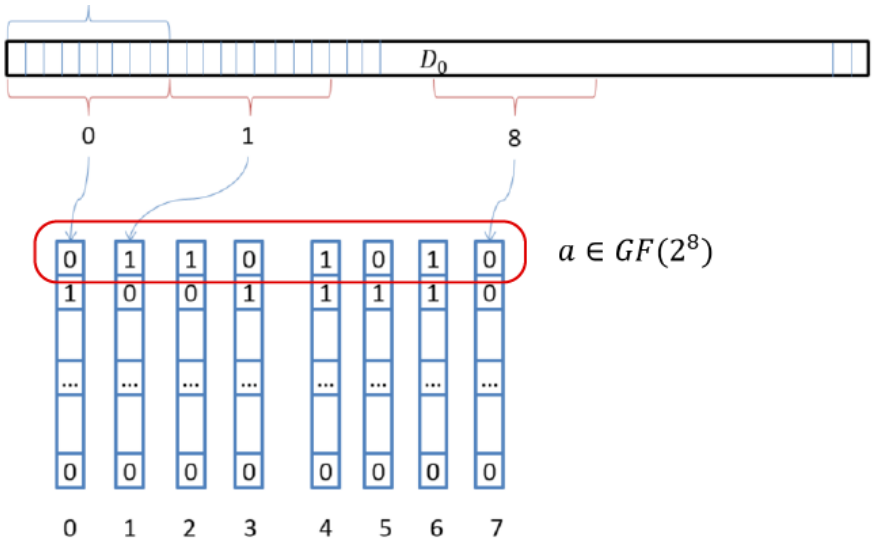

Рис. 3: Вертикальное размещение регистров. Источник: [5]

Плюсы: самый быстрый из представленных, всего 3 операции xor и перестановки для умножения на примитивный элемент поля.
Минусы: для большого числа контрольных сумм требуется больше векторных регистров, требует поддержку векторизации.

4.2. Тестирование

Тестирование проводилось следующим образом:

1. Для выбранного количества страйпов и размера страйпа выделяется область памяти нужного размера, которая впоследствие заполняется случайными данными. Эта память внутри каждого страйпа разделяется на стрипы, которые эмулируют различные жёсткие диски.

2. Запускается функция расчёта синдромов. Вычисленные синдромы сохраняются в соответсвующих стрипах.

3. Проверяется правильность работы функций восстановления утраченных стрипов:

    - Данные страйпа сохраняются в отдельном участке памяти.
    - Случайным образом выбираются номера стрипов, чье повреждение будет эмулироваться.
    - Эти блоки страйпа заполняются нулями.
    - Вызывается функция восстановления поврежденных стрипов, которая использует вычисленные ранее синдромы.
    - Проверяется совпадение данных, сохранённых до повреждения, и тех данных, которые получились после выполнения функции восстановления.

Для измерения времени используется таймер в наносекундах. Для выбранного количества дисков (4, 8, ..., 64) для каждой функции тест проводился 1000 раз, скорость выполнения считалась в мегабайтах в секунду (MB/s). Затем считался доверительный интервал для полученных результатов через медиану и среднеквадратическое отклонение. Для находящихся в доверительном интервале результатов считалось среднее арифметическое значение.

Характеристики тестового сервера для SSE:
ОС: 18.04.2-Ubuntu

CPU: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
RAM: 16 GB

Характеристики тестового сервера (SmartNIC Stingray™ PS1100R) для Neon:
OC: 18.04-Ubuntu
CPU: ARMv8 Cortex-A72 @ 3.0GHz
RAM: 8 GB

5. Результаты

На представленных графиках приведены скорости различных RAID-вычислений для каждого из реализованных алгоритмов.

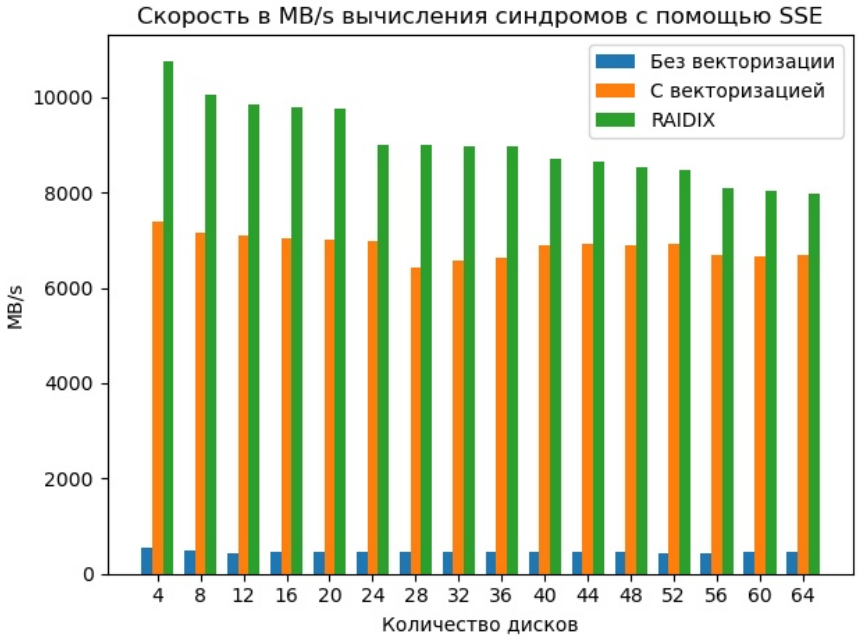

Рис. 4: Вычисление синдромов с помощью SSE

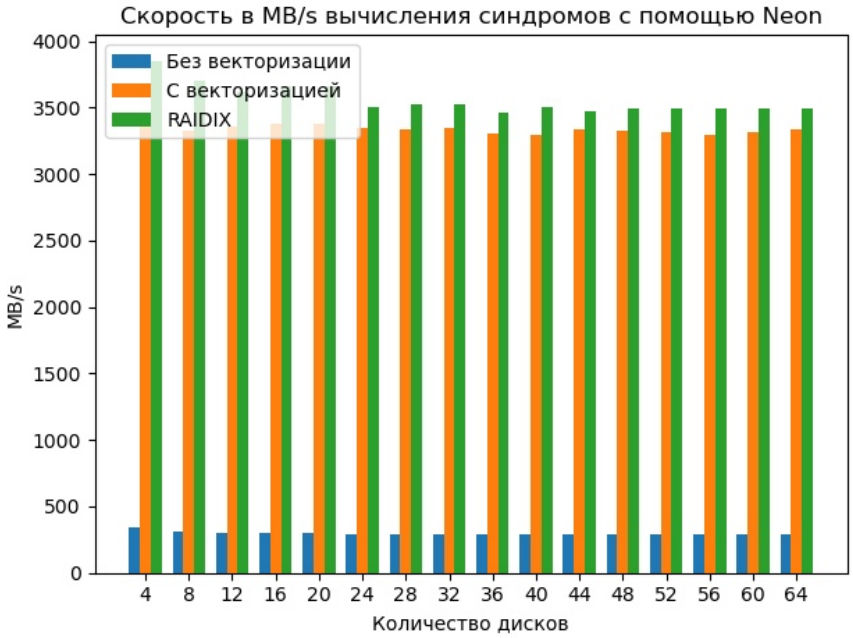

Рис. 5: Вычисление синдромов с помощью Neon

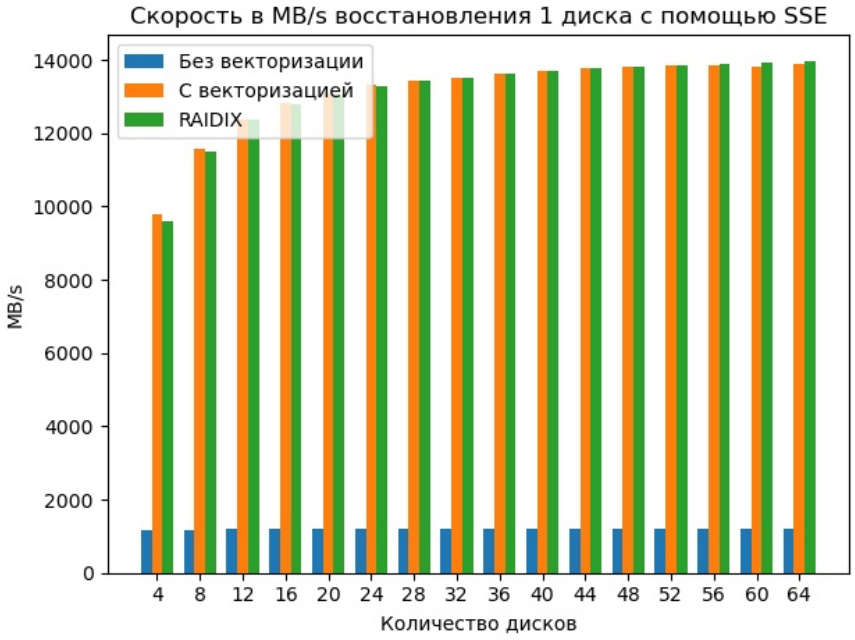

Рис. 6: Восстановление 1 поврежденного диска с помощью SSE

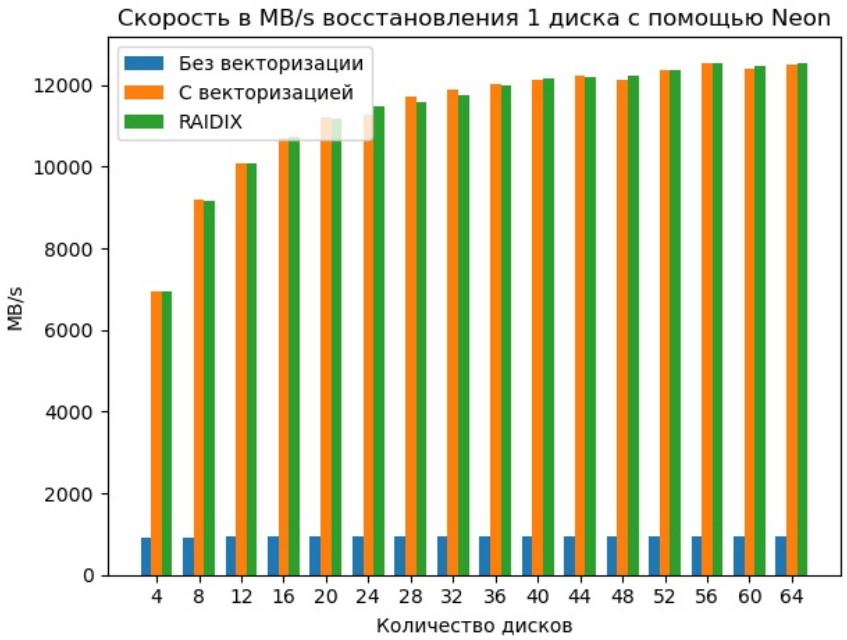

Рис. 7: Восстановление 1 поврежденного диска с помощью Neon

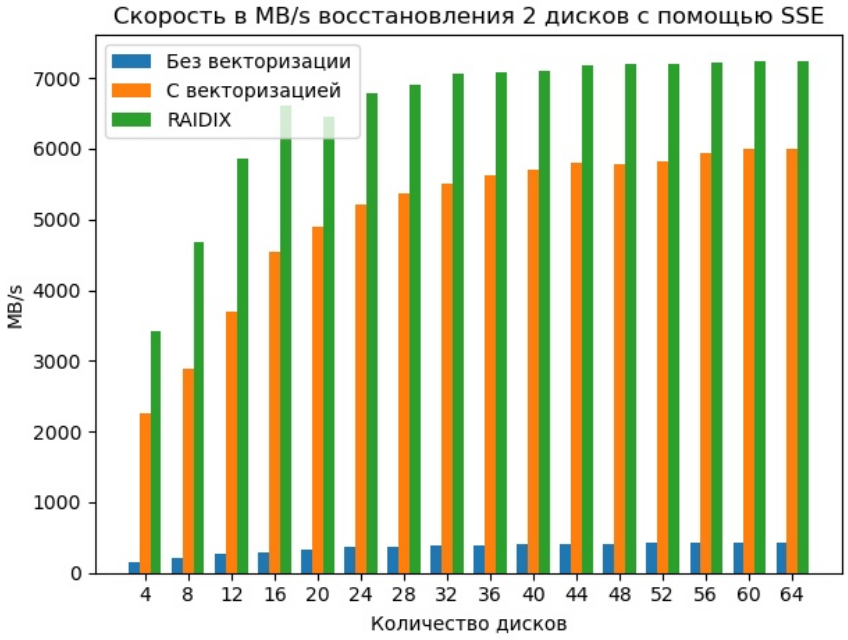

Рис. 8: Восстановление 2 поврежденных дисков с помощью SSE

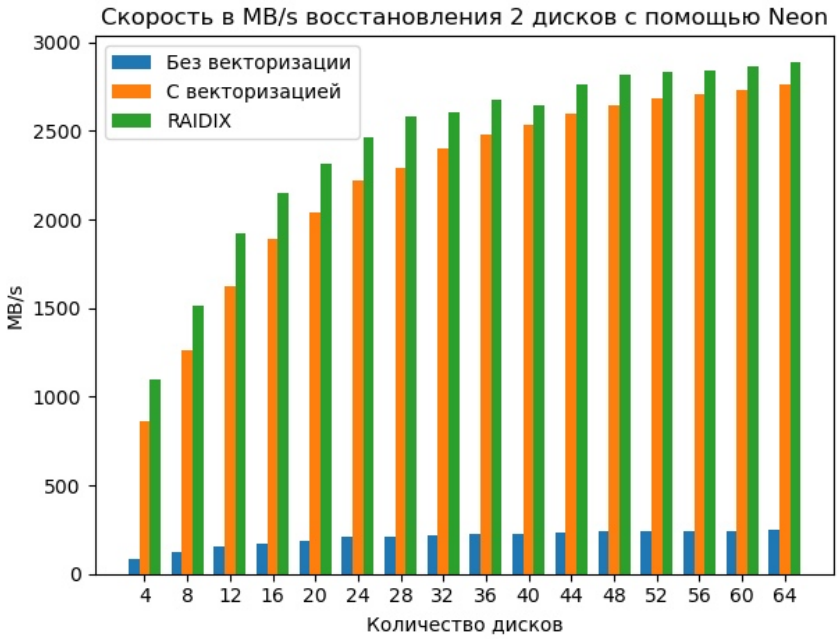

Рис. 9: Восстановление 2 поврежденных дисков с помощью Neon

6. Анализ полученных результатов

Из графиков видно, что метод без векторизации значительно уступает алгоритмам, использующим векторизацию. Например, функция вычисления синдромов выполняется алгоритмом без векторизации примерно в 15 раз медленнее, чем алгоритмом с векторизацией и в 19 раз медленнее, чем алгоритмом RAIDIX с помощью SSE и в 11 и 12 раз медленнее соответственно с помощью Neon.

Восстановление одного поврежденного диска алгоритмами с векторизацией и RAIDIX дает почти одинаковые результаты, так как при восстановлении одного диска используется только синдром $P = \sum_{i=0}^{N-1} D_i$, а следовательно, при восстановленнии нам достаточно только операции XOR (умножение на x не нужно),

то есть реализация этих алгоритов восстановления одного диска не будет отличаться друг от друга.

Алгоритм компании RAIDIX показал самые лучшие результаты. Например, восстановление двух поврежденных дисков происходит в 18 раз быстрее, чем с помощью алгоритма без векторизации и примерно в 1.3 раза быстрее, чем с помощью алгоритма с векторизацией при использовании SSE и в 12 и 1.1 раз соответсвенно при использовании Neon.

Так как производительность SmartNIC Stingray™ PS1100R ниже, чем тестового сервера для SSE, скорости RAID-вычислений на архитектуре ARMv8 оказались ниже.

Заключение

Были реализованы и протестированы 3 алгоритма: без векторизации, с векторизацией и алгоритм компании RAIDIX. Написан краткий обзор предметной области. В SmartNIC перенесены алгоритмы вычисления синдромов и восстановления поврежденных дисков. Проведены сравнительные измерения скоростей реализованных алгоритмов.

В дальнейшем планируется интегрировать полученные алгоритмы векторизации в продукт компании RAIDIX, реализовав их не для одного ядра, а для большего количества (SmartNIC Stingray™ PS1100R имеет 8 ядер), и портировать их в виде модуля ядра.

Список литературы

[1] Broadcom. 2×25-Gb High-Performance Data Center SmartNIC. — 2018. — URL: https://www.broadcom.com/products/ethernet-connectivity/smartnic/ps225 (дата обращения: 14.12.2019).

[2] James S. Plank Kevin M. Greenan Ethan L. Miller. Screaming Fast Galois Field Arithmetic Using Intel SIMD Instructions. — 2013. — URL: http://web.eecs.utk.edu/~jplank/plank/papers/FAST-2013-GF.pdf (дата обращения: 19.05.2020).

[3] Mellanox. BlueField® I/O Processing Unit (IPU). — 2019. — URL: https://www.mellanox.com/related-docs/prod_adapter_cards/PB_BlueField_IPU.pdf (дата обращения: 14.12.2019).

[4] Mellanox. What Is a SmartNIC? — 2019. — URL: https://blog.mellanox.com/2018/08/defining-smartnic/ (дата обращения: 14.12.2019).

[5] RAIDIX. Начнем с математики. Векторизация вычислений в реализации технологии RAID-6. — 2017. — URL: https://m.habr.com/ru/company/raidix/blog/326816/ (дата обращения: 14.12.2019).

[6] Wikipedia. RAID // Википедия, свободная энциклопедия. — 2019. — URL: https://ru.wikipedia.org/wiki/RAID (дата обращения: 14.12.2019).