Инженерный вестник Дона, №12 (2020)
ivdon.ru/ru/magazine/archive/n12y2020/6787

Исследование способов повышения быстродействия кодирования и декодирования кодов Рида-Соломона

С. В. Скороход, А. В. Барлит

Южный федеральный университет, Ростов-на-Дону

Аннотация: Рассматривается задача увеличения скорости кодирования и декодирования кодов Рида-Соломона (RS-кодов). В качестве базовой реализации для сравнения выбрана реализация библиотеки кодирования системы Open JPEG. Данная задача рассматривается для x86-64 окружения. В работе предложены способы повышения производительности как при использовании стандартного набора инструкций, так и при использовании векторных инструкций из наборов SSSE3 и AVX2. Описана методика сравнения быстродействия кодирования/декодирования RS-кодов на примере разработанной библиотеки и базовой реализации в Open JPEG. Проведено экспериментальное исследование взаимосвязи между параметрами RS-кодов, скоростью кодирования/декодирования и набором используемых инструкций. Показано, что для любого RS-кода можно существенно повысить производительность даже на стандартном наборе инструкций. Предложен метод динамического выбора способа ускорения кодирования в зависимости от поддерживаемых целевым компьютером инструкций и параметров используемого RS-кода.
Ключевые слова: помехозащищенное кодирование, коды Рида-Соломона, быстродействие, векторные инструкции.

Введение

Контроль целостности и коррекция ошибок при передаче данных по зашумленным, в частности беспроводным, сетям являются важными задачами, поскольку в процессе передачи информации по таким сетям в блоках данных неизбежно возникают ошибки [1]. Основной стратегией противодействия искажению данных является использование средств упреждающей коррекции ошибок. При этом перед передачей по сети в исходный кодированный поток добавляются избыточные коды четности, которые позволяют произвести коррекцию поврежденных данных принимающей стороной.

Данный подход рассматривается на примере помехозащищённого кодирования потока JPEG 2000, для чего разработана спецификация ITU T.810 [2], в которой описан набор средств защиты, называемых JPWL. Основным средством упреждающей коррекции ошибок в данной

спецификации выступают коды Рида-Соломона (RS-коды). В настоящем исследовании рассматриваются несколько реализаций системы JPWL:

- реализация в составе библиотеки с открытым исходным кодом Open JPEG [3], предназначенная для кодирования изображений в формате JPEG 2000;
- реализация для системы передачи потокового видео, исследуемая в [4].

Обе реализации используют одну и ту же библиотеку для кодирования и декодирования RS-кодов, требования к быстродействию которой особо сильно проявляются в задаче передачи потокового видео, поскольку большая часть времени тратится системой JPWL именно на обработку RS-кодов.

RS-коды — группа кодов, позволяющих исправлять ошибки в блоках данных [5, 6]. Элементами кодового вектора являются группы битов. Все вычисления производятся в полях Галуа (конечных множествах). Вычисления в полях Галуа выполняются с применением полиномиальной арифметики, которая вычислительно сложна при использовании обычных арифметических операций. Коды Рида–Соломона обычно обозначают как $RS(n,k)$, где k – число информационных символов в блоке, n – общее число символов, $t = \lfloor (n-k)/2 \rfloor$ – число случайных ошибок, которые можно исправить в блоке.

Использование различных видов RS-кодов и их влияние на качество декодирования изображений в формате JPEG 2000 при передаче через зашумлённый канал подробно рассмотрено в [7].

Первой целью настоящего исследования является разработка библиотеки ускоренного кодирования и декодирования RS-кодов, содержащей три версии кодера, отличающиеся набором используемых инструкций (исходный код доступен в [8]):

- ALU – используются только стандартные процессорные операции;

- SSSE3, AVX2 – используются соответствующие расширенные наборы векторных инструкций процессора (архитектура x86-64).

Второй целью исследования является проведение экспериментального сравнения быстродействия разработанной библиотеки и базовой реализации в Open JPEG, позволяющего выявить характер взаимосвязи между параметрами RS-кодов, скоростью обработки данных и набором используемых инструкций.

В настоящем исследовании также предлагается решить задачу динамического выбора наиболее эффективной версии кодера на основе полученных результатов сравнения быстродействия.

Способы повышения производительности

В реализации Open JPEG используются известные алгоритмы кодирования и декодирования. Вычисления в полях Галуа выполняются с применением таблицы умножения, медленная операция вычисления целочисленного остатка от деления заменена на более эффективную аналогичную функцию.

Если сравнить разработанную версию ALU с реализацией в Open JPEG, то можно выделить следующие способы оптимизации «горячих» участков кода, которые позволили получить повышение скорости обработки данных:

- оптимизация функции вычисления остатка от целочисленного деления;
- оптимизация таблицы для умножения чисел в поле Галуа, позволяющая исключить лишние проверки во время вычислений.

Например, при декодировании всегда выполняется операция вычисления синдромов [5], при этом требуется выполнить $n * (n - k)$ раз вложенный цикл в блоке кода, приведённом в таблице 1 (для сравнения приведены блоки кода из Open JPEG и разработанного кодера версии ALU).

Очевидно, что любые дополнительные действия во вложенном цикле существенно влияют на быстродействие. Так, в версии Open JPEG первая операция – проверка условия, не является ли символ «запрещённым» для

просмотра по таблице умножения (умножение на 0). При просмотре по таблице умножения выполняется вычисление остатка от деления на 255, соответствующая функция «modnn» в Open JPEG содержит уже не просто проверку условия, а цикл.

Таблица № 1

Вычисление синдромов

Open JPEG	ALU

`static int modnn(int x) {`
`  while (x >= 255) {`
`  x -= 255;`
`  x = (x >> 8) + (x & 255);`
`  }`
`  return x;`
`}`
`for (i = 1; i <= n - k; i++) {`
`  tmp = 0;`
`  for (j = 0; j < n; j++) {`
`  if (recd[j] != 255)`
`  tmp^=Alpha_to[modnn(recd[j]+(B0+i-1)*j)];`
`  }`
`  syn_error |= tmp;`
`  s[i] = Index_of[tmp];`
`}`
	`static void inline mod255(int* x) {`
	`  *x = (*x >> 8) + (*x & 0xff);`
	`  *x = (*x >> 8) + (*x & 0xff);`
	`}`
	`for (int j = 0; j < n - k; j++) {`
	`  int v = n - 1, s = 0;`
	`  for (int m = 0; m <= v; m++) {`
	`  int ecx = (v - m) * j;`
	`  mod255(&ecx);`
	`  s ^= lutExp[stemp[m] + ecx];`
	`  }`
	`  hasErrors |= s;`
	`  syn[j] = lutLog[s];`
	`}`

В кодере версии ALU реализованы оптимизации указанных проверок условий. Оптимизированная функция вычисления остатка от деления «mod255» построена на следующих соображениях: максимальное число, от которого будет вычисляться остаток, равно $2^{16}-1$, следовательно, цикл в функции «modnn» будет выполнен не более 2 раз. От вычитания константы 255 можно также отказаться, но в некоторых случаях это может привести к тому, что остаток будет равен 256. Эта проблема легко решается путём небольшой модификации таблицы умножения. Оптимизация таблицы умножения также включает в себя расширение её размера так, чтобы можно было осуществлять поиск «запрещённого» символа.

Предложенные способы позволили исключить проверки условий на «горячем» пути кода, что можно увидеть в столбце ALU таблицы 1.

Использование векторных инструкций позволяет ещё больше ускорить вычисления. В [9] приведён способ очень быстрого умножения типа «вектор на скаляр». Основу этому способу составляет инструкция процессора «pshufb xmm, xmm» из набора SSSE3, которая рассматривает первый регистр xmm как массив из 16 индексов, а второй как таблицу из 16 байт, из которой производится подстановка байт по индексам в регистр результата. По сути, это такое же умножение по таблице, как и в версии ALU, но в таблице может быть максимум 16 элементов. Чтобы выполнить умножение по таблице из 16 элементов, умножение двух чисел представляется как сумма двух произведений: на скаляр: по отдельности умножаются младшие и старшие 4 бита каждого из 16 байт.

Дополнительно требуется свести участки алгоритма, в которых производится умножение чисел, к использованию умножения «вектор на скаляр». Это требование обосновано тем, что умножение «вектор на вектор» для 8-битных символов нельзя реализовать так же эффективно.

Векторные инструкции в языке C представлены в виде специальных функций, описание которых доступно в [10]. В режиме компиляции Release выполняется автоматическое распределение доступных регистров, что делает написанный код максимально эффективным по быстродействию.

Таблица 2 приведена для демонстрации отличий векторной версии алгоритма от версии ALU, приведенной в таблице 1.

Несмотря на то, что в сумме получается заметно больше операций, чем в версии ALU из таблицы 1, версия SSSE3 всё равно выигрывает за счёт того, что за одну итерацию вложенного цикла вычисляются сразу 16 значений, и так же нет проверок условий.

Набор инструкций AVX2 использует векторы размером 256 бит, и для RS-кодов с 16 или менее контрольными символами 256 бит вектор будет занят не более чем на половину объёма. Основным преимуществом набора инструкций AVX является в 2 раза увеличенный размер векторов

относительно инструкций SSE, поэтому в случае с RS-кодом (48,32) с 16 контрольными символами производительность кодера AVX2 не может превышать производительность кодера SSSE3, что будет показано в результатах экспериментального сравнения.

Таблица № 2

Использование инструкций SSSE3

Умножение вектора на скаляр	Вычисление синдромов

`static void inline GF_mvs_SSSE3(__m128i* vtt, __m128i* vio, __m128i* vmask, __m128i* lmul, __m128i* umul) {`
`  *vtt = _mm_and_si128(*vio, *vmask);`
`  *vtt = _mm_shuffle_epi8(*lmul, *vtt);`
`  *vio = _mm_srli_epi64(*vio, 4);`
`  *vio = _mm_and_si128(*vio, *vmask);`
`  *vio = _mm_shuffle_epi8(*umul, *vio);`
`  *vio = _mm_xor_si128(*vio, *vtt);`
`}`
	`int steps = (n - k - 1) >> 4;`
	`for (int j = n - 2; j >= 0; j--) {`
	`  int idx = buffer[j] * 32;`
	`  __m128i* lutv = (__m128i*)(lutSSE + idx);`
	`  __m128i umul = _mm_load_si128(lutv);`
	`  __m128i lmul = _mm_load_si128(lutv + 1);`
	`  __m128i* lr = (__m128i*)roots;`
	`  for (int m = 0; m <= steps; m++) {`
	`  __m128i vio = _mm_load_si128(lr++), vt;`
	`  GF_mvs_SSSE3(&vt, &vio, &vmask, &lmul, &umul);`
	`  vt = _mm_load_si128(ls);`
	`  vt = _mm_xor_si128(vt, vio);`
	`  _mm_store_si128(ls++, vt);`
	`  }`
	`  ls = (__m128i*)syn;`
	`  roots += 256; //Hard to explain`
	`}`

Методика исследования

При использовании зашумлённой IP сети может произойти искажение как данных пакета, так и заголовка пакета, результатом которого является потеря пакета и всех содержащихся в нём данных. Исходя из этого, целесообразным является кодирование блоков данных с чередованием при размещении их в пакетах. В таком случае потеря пакетов приводит к появлению ошибок в нескольких закодированных блоках данных, и потерянные пакеты можно таким образом восстановить [11].

В данном исследовании используются следующие допущения:

- чередование данных не влияет на скорость кодирования и декодирования;
- при использовании RS-кода с мощностью t число ошибок в блоке не превосходит t.

Для исследования используется консольное приложение, написанное на языке C++ в среде MS VS 2019. Рабочее окружение: процессор Intel® Broadwell-EP с частотой 4.2GHz, ОЗУ 32Гб, компилятор MSVC (Microsoft Visual Studio 2019), ОС Windows 10 Pro 1909.

В исследовании участвуют 4 кодера:

- JPWL – встроенная реализация RS-кодов из библиотеки Open JPEG;
- ALU, SSSE3, AVX2 – варианты разработанной библиотеки.

Для тестирования используется любой файл достаточно большого размера, чтобы время его декодирования составляло не менее 0.2с для всех тестируемых RS-кодов и всех вариантов разработанной библиотеки (отличающихся набором используемых инструкций). Результаты тестирования выводятся в форматированный файл, который затем загружается в табличный редактор для нормализации и визуализации.

Абсолютное значение скорости кодирования (например, в Мб/с) недостаточно наглядно, т.к. оно отличается в несколько раз для RS-кодов с малой и большой избыточностью. Также оно существенно зависит от процессора, на котором выполняются тесты. Исходя из этих соображений, результаты тестирования нормализуются, за 1 принимается скорость обработки данных встроенным в Open JPEG кодером. Пусть скорость обработки с помощью Open JPEG равна s_j, а скорость обработки исследуемого кодера равна s^*, тогда нормализованный результат исследуемого кодера s^*_{norm} определяется как $s^*_{norm} = s^*/s_j$.

Переменными параметрами исследования являются применяемые RS-коды и наличие или отсутствие ошибок в блоках данных. Программа исследования приведена в таблице 3.

Таблица № 3

Программа исследования

Тестируемые RS-коды	Используемые кодеры	Измерения

RS(48,32), RS(64,32), RS(96,32), RS(128,32)	JPWL, ALU, SSSE3, AVX2	Время кодирования, время декодирования данных без ошибок и с ошибками

Исследовательский программный комплекс

Для проведения исследования был разработан программный комплекс, схема функционирования которого изображена на рис. 1.

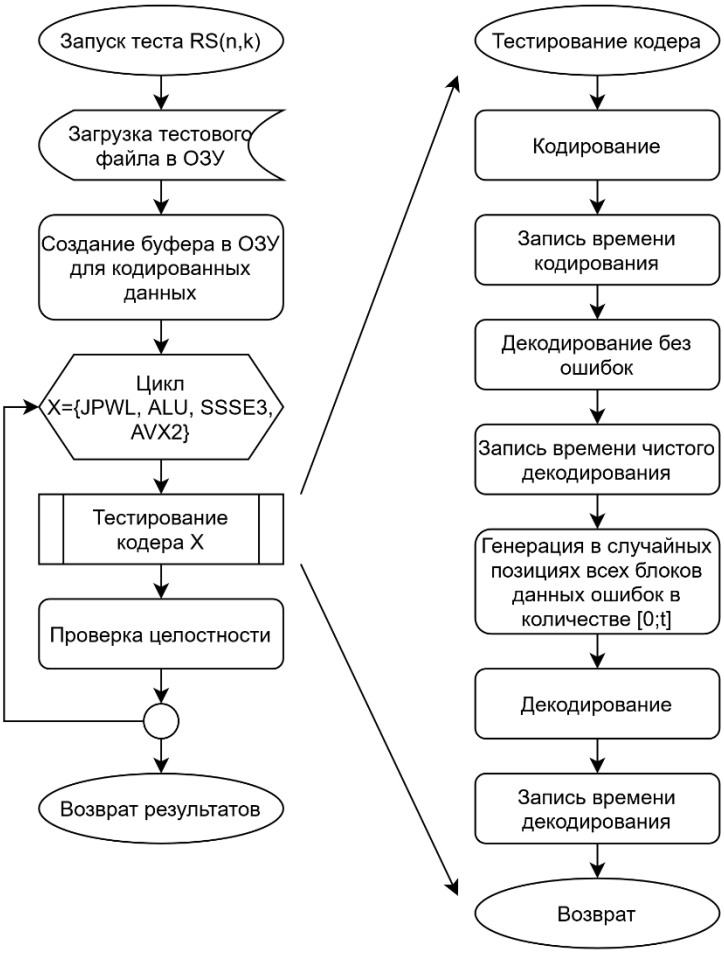

Рис. 1. – Схема работы исследовательского программного комплекса

В качестве входных данных для работы программного комплекса используются тестовый файл и исследуемый RS-код. Комплекс может эксплуатироваться в пакетном режиме, что позволяет протестировать все нужные RS-коды.

На подготовительном этапе тестовый файл загружается в буфер в ОЗУ, и дополнительно создаются два буфера: для хранения результата кодирования файла и результата последующих декодирований. Полученные три буфера используются далее при тестировании.

На этапе тестирования запускается последовательно один и тот же набор тестов для всех кодеров. Время выполнения каждой операции тестирования измеряется с помощью вызовов системной функции, возвращающей более точное для исследования процессорное время.

Первой операцией тестирования выступает кодирование исходного файла. Исходный буфер представляется как последовательно записанные блоки данных, которые по очереди кодируются и так же последовательно записываются в буфер для результата кодирования.

Вторая операция тестирования – декодирование полученного буфера (т.к. в буфере нет ошибок, то операция для ясности будет называться «чистым декодированием»). Буфер с результатом кодирования подаётся в виде блоков декодеру, который записывает данные в буфер для результата декодирования. Далее для контроля целостности проводится сравнение буфера с декодированными данными и исходного буфера.

Третья операция тестирования – декодирование буфера, содержащего ошибки. Для внесения ошибок в буфер используется генератор псевдослучайных последовательностей чисел. Буфер представляется как набор блоков, и для каждого блока генерируется число ошибок в диапазоне $[0;t]$, где t – мощность исследуемого RS-кода, таким образом среднее число ошибок будет равно $t/2$. Далее в цикле генерируются позиции ошибок в блоке и ненулевые значения ошибок, которые накладываются на исходные данные путём операции XOR. Очевидно, что количество сгенерированных таким образом ошибок при этом может не совпадать с заданным, т.к. генерируемые позиции внутри блока могут повторяться, однако на большом

числе блоков среднее количество ошибок в блоке всё так же стремится к $t/2$. Буфер с ошибками подаётся в виде блоков декодеру, который записывает данные в буфер для результата декодирования. Далее для контроля целостности проводится сравнение буфера с декодированными данными и исходного буфера.

Результатом моделирования выступает статистика, которая для каждого эксперимента (каждой пары значений используемого RS-кода и тестируемого кодера) включает в себя три значения (в Мб/с): скорость кодирования, скорость «чистого» декодирования и скорость декодирования данных с ошибками. В данном исследовании для каждого эксперимента в качестве тестового файла использовалось изображение объёмом 9.9Мб.

Результаты тестирования

На рис. 2 приведён график экспериментальной зависимости относительной производительности кодирования данных для всех сочетаний кодеров и RS-кодов из таблицы 3.

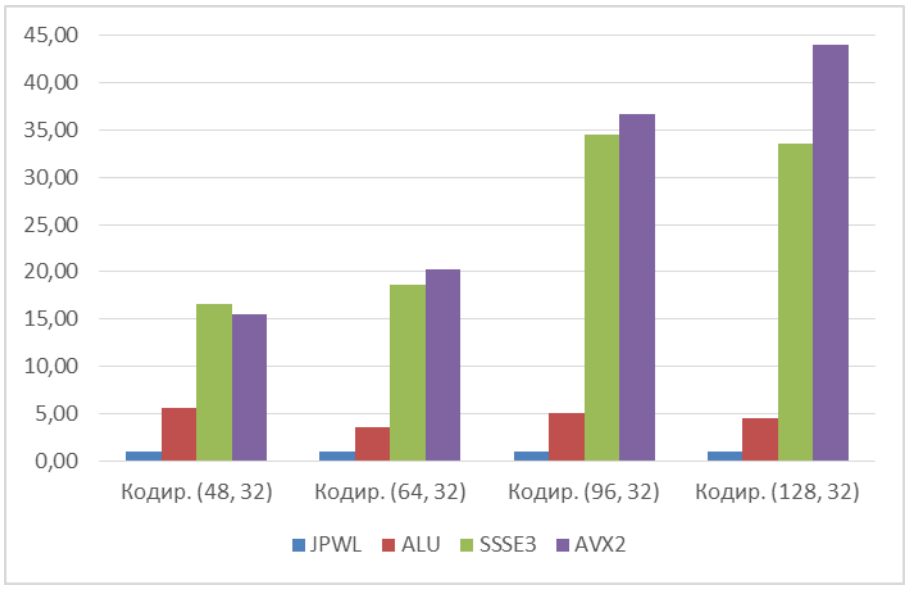

Рис. 2. – Относительная производительность кодирования

При усреднении показателей относительной производительности всех RS-кодов из анализа рис. 2 получаем, что версия ALU быстрее JPWL в 4.6

раза, версия SSSE3 быстрее в 25.8 раза, версия AVX2 быстрее в 29.1 раза.

На рис. 3 приведён график экспериментальной зависимости относительной производительности декодирования данных без ошибок для всех сочетаний кодеров и RS-кодов из таблицы 3.

При усреднении показателей относительной производительности всех RS-кодов из анализа рис. 3 получаем, что версия ALU быстрее JPWL в 3.4 раза, версия SSSE3 быстрее в 35 раз, версия AVX2 быстрее в 49.9 раза.

Рис. 3. – Относительная производительность чистого декодирования

На рис. 4 приведён график экспериментальной зависимости относительной производительности декодирования данных с ошибками для всех сочетаний кодеров и RS-кодов из таблицы 3:

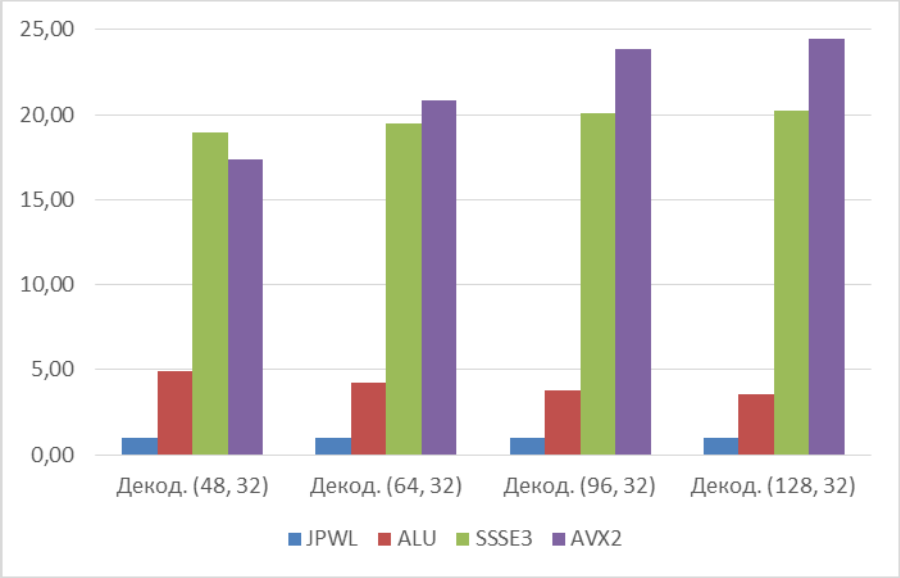

Рис. 4. – Относительная производительность декодирования

При усреднении показателей относительной производительности всех RS-кодов из рис. 4 получаем, что версия ALU быстрее JPWL в 4.1 раза, версия SSSE3 быстрее в 19.7 раза, версия AVX2 быстрее в 21.6 раза.

Заключение

Полученные результаты экспериментального исследования показывают, что даже без использования векторных инструкций можно существенно повысить производительность такой ресурсоёмкой операции, как декодирование данных с ошибками.

При операции декодирования данных с ошибками (рис. 4) кодеры SSSE3 и AVX2 показывают меньший относительный прирост производительности, чем для кодирования данных и декодирования данных без ошибок. Это объясняется тем, что алгоритм декодирования данных с ошибками помимо вычисления синдромов содержит ещё 4 последовательных преобразования, в которых участки «горячего кода» более объёмные, и применение векторных инструкций оказывает меньшее влияние на время выполнения.

На основе результатов сравнения различных кодеров предлагается

следующий алгоритм динамического выбора версии кодера для кодов $RS(n,k)$:

- если процессор поддерживает набор инструкций AVX2 и $n-k>16$, то выбирается версия AVX2;
- если $n-k\leq 16$ или процессор поддерживает только набор инструкций SSSE3, выбирается версия SSSE3;
- во всех остальных случаях выбирается версия ALU.

Литература

1. Альбекова З. М., Квашурин В. О., Тутик Н. А. Анализ эволюции технологии беспроводных сетей и прогнозы развития инфокоммуникационных сетей в России // Инженерный вестник Дона, 2016, №4. URL: ivdon.ru/ru/magazine/archive/n4y2016/3933.

2. ITU-T Recommendation T.810. Information Technology JPEG2000 Image Coding System: Wireless. Geneva: ITU, 2007. 60 p.

3. Official repository of the OpenJPEG project. URL: github.com/uclouvain/openjpeg (дата обращения 24.08.2020)

4. Скороход С. В., Скороход Д. С. Оценка интенсивности зашумления в канале при передаче изображений в формате JPWL на основе экспериментальной модели // Инженерный вестник Дона, 2018, №4. URL: ivdon.ru/ru/magazine/archive/n4y2018/5305

5. Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение / пер. с англ. Афанасьева В. Б. М.: Техносфера, 2006. 320 с.

6. Серина М. С., Малыгин Д. Ю., Скворцов М. В. Декодирование кодов Рида-Соломона // Алгоритмы, методы и системы обработки данных, 2004, №9-2. URL: elibrary.ru/download/elibrary_17421443_75802285.pdf.

7. Скороход С. В., Дроздов С. Н., Скороход Д. С. Исследование зависимости качества декодированного изображения в формате JPEG 2000 от

параметров JPWL и частоты пакетных ошибок в зашумленном канале // Инженерный вестник Дона, 2017, №4. URL: ivdon.ru/ru/magazine/archive/n4y2017/4453.

8. Reed-Solomon library. URL: github.com/Botinok666/RS (дата обращения 28.08.2020)

9. Plank J. S., Greenan K. M., Miller E. L. Screaming Fast Galois Field Arithmetic Using Intel SIMD Instructions // 11th USENIX Conference on File and Storage Technologies. URL: ssrc.ucsc.edu/Papers/plank-fast13.pdf (дата обращения 13.08.2020)

10. The Intel Intrinsics Guide. URL: software.intel.com/sites/landingpage/IntrinsicsGuide/ (дата обращения 01.09.2020)

11. Скороход С. В., Хусаинов Н. Ш. Исследование средств JPWL в условиях коррекции пакетных ошибок при передаче видео в формате JPEG 2000 // Известия ЮФУ. Технические науки. 2016, №8 (181). С. 14–26.

References

1. Al'bekova Z. M., Kvashurin V. O., Tutik N. A. Inzhenernyj vestnik Dona, 2016, №4. URL: ivdon.ru/ru/magazine/archive/n4y2016/3933.

2. ITU-T Recommendation T.810. Information Technology JPEG2000 Image Coding System: Wireless. Geneva: ITU, 2007. 60 p.

3. Official repository of the OpenJPEG project. URL: github.com/uclouvain/openjpeg (access date 24.08.2020)

4. Skorokhod S.V., Skorokhod D.S. Inzhenernyj vestnik Dona, 2018, №4. URL: ivdon.ru/ru/magazine/archive/n4y2018/5305

5. Morelos-Saragosa R. Iskusstvo pomekhoustojchivogo kodirovaniya. Metody, algoritmy, primenenie [The art of error-correcting coding. Methods, algorithms, application]. Perevod s anglijskogo Afanasyev V. B. M.: Tehnosphera, 2006. 320 p.

6. Serina M. S., Malugin D. Y., Skvortsov M. V. Algoritmy, metody i sistemy obrabotki dannyh, 2004, №9-2. URL: elibrary.ru/download/elibrary_17421443_75802285.pdf.

7. Skorohod S. V., Drozdov S. N., Skorohod D. S. Inzhenernyj vestnik Dona, 2017, №4. URL: ivdon.ru/ru/magazine/archive/n4y2017/4453.

8. Reed-Solomon library. URL: github.com/Botinok666/RS (access date 28.08.2020)

9. Plank J. S., Greenan K. M., Miller E. L. 11th USENIX Conference on File and Storage Technologies. URL: ssrc.ucsc.edu/Papers/plank-fast13.pdf (access date 13.08.2020)

10. The Intel Intrinsics Guide. URL: software.intel.com/sites/landingpage/IntrinsicsGuide/ (access date 01.09.2020)

11. Skorohod S. V., Khusainov N. S. Izvestiya YuFU. Tekhnicheskie nauki. 2016, №8 (181). pp. 14–26.