УДК 004.62

Винников В.В.[1,2], Иваничкина Л.В.[3,4]

[1] ООО «Акронис», г. Москва, Россия
[2] Вычислительный центр им. А.А. Дородницына ФИЦ ИУ РАН, г. Москва, Россия
[3] ООО «Проект Икс», г. Москва, Россия
[4] Московский Физико-технический институт, МФТИ, г. Долгопрудный, Россия

ПРОГРАММНО-АЛГОРИТМИЧЕСКАЯ РЕАЛИЗАЦИЯ КОДОВ РИДА-СОЛОМОНА ДЛЯ ПОЛЕЙ ГАЛУА ВЫСОКОГО ПОРЯДКА

АННОТАЦИЯ

В статье приводится оригинальный подход к программно-алгоритмической реализации кодов Рида—Соломона для систем хранения сверхбольших объемов данных. Сравниваются самые популярные алгоритмы алгебраических операций умножения над конечными полями, рассматриваются их преимущества и недостатки, зависящие от мощности алфавитов кодируемых сообщений. В работе показаны различия в требованиях к параметрам избыточного кодирования для систем помехоустойчивой передачи данных по каналам связи и систем долгосрочного надёжного хранения данных с отказоустойчивым доступом. Предложен алгоритм умножения, в котором подходы к быстрому умножению с помощью таблиц и методов «разделяй и властвуй» скомбинированы с целью достижения компромисса между объемом выделяемой оперативной памяти и количеством затрачиваемых арифметических операций. Приведено описание программной реализации алгоритма на языке C++ в виде листингов.

КЛЮЧЕВЫЕ СЛОВА

Конечные поля; поле Галуа; код Рида—Соломона, генерирующая матрица; матрица Вандермонда, алгоритм Монтгомери, алгоритм Карацубы.

Vinnikov V.V.[1,2], Ivanichkina L.V.[3,4]

[1] OOO Acronis, Moscow, Russia
[2] Dorodnicyn Computing Centre of RAS, Moscow, Russia
[3] OOO Project Iks, Moscow, Russia
[4] Moscow Institute of Physics and Technology, Dolgoprudny, Russia

ALGORITHMIC AND SOFTWARE IMPLEMENTATION OF REED–SOLOMON ERASURE CODES FOR GALOIS FIELDS OF HIGH ORDER

ABSTRACT

This paper is concerned with the novel approach to algorithmic and software implementation of Reed–Solomon codes for super large data storage systems. We compare the most popular algorithms of multiplication over finite fields and consider their pros and cons depending on the alphabet power of messages to encode. This work emphasizes differences in the redundancy parameter requirements for erasure codes designed for either noise-resistant data transfer over communication channels or long term reliable data storing with fault-tolerant access. We propose the multiplication algorithm that combines table and "divide and conquer" approaches to fast multiplication to achieve a compromise between the volume of allocated random access memory and the number of required arithmetic operations. The paper provides the listings with C++ program implementation of the presented algorithm.

KEYWORDS

Finite fields; Galois Field; Reed–Solomon code; generator matrix; Vandermonde matrix, Montgomery multiplication, Karacuba multiplication.

Современные системы хранения данных (далее СХД) петабайтного объема включают в свой состав тысячи дисковых накопителей. Подобная крупная популяция элементов подчиняется законам математической статистики, а, следовательно, определённая доля дисков подвержена регулярному выходу из строя [1]. При этом, совокупная частота единичных и коллективных отказов растёт с увеличением мощности множества задействованных жестких дисков. Предотвращение

необратимой потери данных достигается резервированием содержимого СХД с помощью добавления некоторой избыточности в исходный набор данных. Распространены два варианта построения избыточности: резервное копирование (репликация) и помехоустойчивое кодирование блоков, занимающее меньше избыточного дискового пространства. Уровень надёжности хранения определяется количеством полных копий (реплик) или параметрами кодирования. На практике широко используются (n, k)-коды Рида-Соломона (см, например, [2]), в которых входное сообщение из k символов алфавита мощностью $q = 2^{\omega}$ преобразуется в n символов из того же алфавита. Коды Рида—Соломона вносят минимально возможный объем избыточности и обеспечивают максимальный уровень помехоустойчивости.

При заданных значениях параметров их разность $(n - k)$ определяет максимальное количество утраченных или искаженных символов, при котором из неповреждённых символов может быть восстановлено исходное сообщение. Подобное помехоустойчивое представление данных используется как при передаче, так и при хранении информационных сообщений. При этом, потоковая передача данных по каналам связи и блочное размещение данных на накопителях предъявляют различные требования к параметрам кодирования n и k.

Так, для потоковой передачи данных при внесении в сообщение пакета ошибок незначительно искажается серия информационных символов с неизвестным расположением, что требует большой величины $(n - k)$ и алфавита малой мощности $q{\sim}2^8$, соответствующего однобайтовой (восьмибитной) структуре данных. Кроме того, с целью сохранения эффективной пропускной способности канала необходимо ограничить накладные расходы на избыточность вида (n/k). Эти факторы определяют коды, общепринятые в коммуникационной отрасли, например, стандарт DVB (Digital Video Broadcast) [3] с параметрами $n = 204$ и $k = 188$. Дополнительно, локализация априори неизвестных ошибок требует применения специальных вычислительно ёмких алгоритмов поиска искаженных символов внутри сообщения.

В свою очередь, надёжное помехоустойчивое хранение данных требует размещения каждого символа (фрагмента) из закодированного сообщения (блока данных) на отдельном накопителе СХД, независимом от других устройств хранения. В случае аппаратного отказа одного или нескольких дисков такая схема распределения данных приводит к необходимости обращения к априори известным исправным дискам при восстановлении блока и передачи доступных фрагментов по сети в узел декодирования. Поскольку каждое считывание и передача фрагмента затрачивают полезные ресурсы системы, количество подобных действий необходимо сократить, то есть использовать (n, k)-код с малыми значениями параметров k и n. При этом, для поддержания высокой производительности требуется сохранить битовую длину исходного сообщения за счёт увеличения битовой длины фрагмента данных (мощности алфавита) до значений $q{\sim}2^{64}$ и выше.

Следует отметить, что типовая вычислительная техника ограничена машинной арифметикой над целыми 64-разрядными числами. Согласно теории конечных полей Галуа, вычислительная сложность алгебраических операций по модулю w значительно возрастает с увеличением w из-за согласования переносов разрядов между машинными словами, составляющими фрагмент блока данных. Растут также и затраты на хранение в памяти промежуточных результатов расчётов. Тем не менее, совершенствование вычислительной техники, развитие теоретической и прикладной криптографии, а также накопление результатов численных экспериментов по конструктивному описанию полей Галуа высокого порядка [4], дают возможность программно реализовать алгоритмы помехоустойчивого кодирования по схемам Рида—Соломона для фрагментов данных большого объема.

Согласно теории, коды Рида—Соломона одновременно принадлежат классам линейных циклических блочных и полиномиальных кодов. Исходя из этого, представление кодов Рида—Соломона допускает две общепринятые интерпретации. В первой трактовке используется таблица многочленов степени меньше k над конечным полем порядка n, где n — степень простого числа. В процессе кодирования k входных символов рассматриваются как первый сегмент таблицы многочленов степени меньше k. Теорией гарантируется, что в таблице содержится единственный многочлен, соответствующий этим k символам. Оставшиеся $(n - k)$ символов определяются как значения этого многочлена в соответствующих $(n - k)$ целочисленных точках. В этой схеме избыточность достигается за счёт переопределенности системы, в которой количество уравнений превышает количество искомых переменных. На практике применяется более производительный способ кодирования, когда k входных символов считаются коэффициентами многочлена p_x степени меньше k, а дополнительные $(n - k)$ символов являются полиномиальными коэффициентами произведения p_x на циклический генерирующий многочлен, определяемый из таблиц неприводимых многочленов, например, [4]. Во второй трактовке вместо генерирующего многочлена используется генерирующая матрица размерностью $k{\times}n$. Различают также

систематический и несистематический варианты кодов. В систематическом коде символы избыточности дописываются в конец исходного сообщения. Левая квадратная часть генерирующей матрицы в этом случае представляет собой единичную матрицу. В несистематическом коде, исходя из структуры генерирующей матрицы, итоговое закодированное сообщение может не содержать ни одного символа исходного сообщения. К преимуществам систематического кода относится тривиальное декодирование сообщения при отсутствии ошибок, а недостаток заключается в необходимости вычисления обратных матриц. Несистематическое кодирование требует нетривиального декодирования даже неискаженного сообщения, однако позволяет использовать легкообращаемые матрицы, например, инволютивные или ортогональные.

В настоящей работе приведены примеры алгоритмической и программной реализации алгебраических операций, используемых в систематическом коде Рида—Соломона с генерирующей матрицей Вандермонда. Для программной реализации этого кода на конечном поле Галуа с алфавитом мощностью $q = 2^{\omega}, \omega > 64$ были построены и запрограммированы алгоритмы сложения и умножения больших чисел по модулю 2^{ω}. На основе этих алгоритмов стоится генерирующая матрица размерностью $k \times n$ символов и производится кодирование путём умножения вектор-строки исходного сообщения из k символов на эту матрицу. Таким же образом реализовано декодирование, а именно: исключение из прямоугольной матрицы $(n - k)$ столбцов, соответствующих исключаемым позициям контрольных или утраченных символов в закодированном сообщении длины n, с последующим обращением полученной квадратной матрицы $k \times k$ и умножением сокращенной закодированной вектор-строки длины k на неё.

Основная вычислительная сложность всех алгоритмов кодирования с высокой мощностью алфавита $q = 2^{\omega}$, $\omega > 64$ вызвана вычислительными затратами на выполнение операций умножения над многобитовыми числами, представленными последовательностью машинных слов. Все известные методы перемножения двух больших чисел могут быть классифицированы по количеству затрачиваемых машинных операций. Так, классический метод умножения в двоичной системе счисления является наиболее затратным, несмотря на высокую скорость проведения индивидуальных операций сложения и битового сдвига машинных слов.

В наиболее производительном методе используется плоская таблица с результатами перемножения двух операндов. При условии, что таблица вычисляется однократно, метод будет задействовать лишь быстрые операции адресного считывания в памяти. Несмотря на простоту реализации, этот метод является исключительно требовательным к дорогостоящей части выделяемых аппаратных ресурсов, поскольку таблица занимает $2^{\omega} \times 2^{\omega} \times 2\omega$ бит в быстрой оперативной памяти. При мощности алфавита $q = 2^8$ занимаемый таблицей объем составит 128 КиБ, однако, при удвоении разрядности ω до мощности алфавита $q = 2^{16}$ в памяти потребуется выделить уже 8 ГиБ. Таким образом, прямое использование таблиц умножения для чисел разрядностью $\omega = 32$ и выше, не представляется оправданным.

Используемые на практике методы перемножения длинных операндов построены на принципе «разделяй и властвуй». В основу этого принципа положена возможность представить множитель как сумму, в которой каждое слагаемое занимает свой диапазон разрядов. В этом случае искомое произведение может быть вычислено с использованием произведений слагаемых каждого из исходных операндов. При этом, при перемножении слагаемых меньшей разрядности часто используется рекурсивный вызов алгоритма до достижения разрядности машинной арифметики. Первым представителем этого семейства методов является алгоритм Карацубы [5] с использованием двух слагаемых на операнд. В развитие этого метода были предложены обобщающие методы с многократным расщеплением, такие как метод Тума—Кука [6]. Академическое признание также получили методы умножения, использующие преобразование Фурье, такие как алгоритмы Шёнхаге — Штрассена [7] и Фюрера [8]. Эти методы имеют наименьшую асимптотическую алгоритмическую сложность, однако, превосходство над алгоритмом Тума—Кука достигается лишь, начиная с чисел порядка $2^{2^{15}}$—$2^{2^{17}}$, что существенно ограничивает область практического применения. Кроме того, эффективная программная реализация методов на основе преобразования Фурье требует перехода от арифметики комплексных чисел к специальному теоретико-числовому преобразованию (Number-theoretic transform) в конечном поле.

В своих первоначальных вариантах большинство методов умножения больших чисел используют арифметику на множестве целых чисел. Исходя из этого, непосредственное использование методов типа Тума–Кука в алгоритмах кодирования в значительной степени ограничено и допустимо лишь на промежуточных уровнях рекурсии, не требующих операции взятия остатка по модулю. Иными словами, результат перемножения двух операндов разрядностью ω лежит в пределах разрядности 2ω, что влечёт необходимость дополнительного деления

результата по модулю 2^{ω}. Эта дополнительная операция в случае массового применения также является вычислительно затратной, поэтому на практике в качестве метода умножения по модулю 2^{ω} используется метод Монтгомери [9].

В методе Монтгомери символ a алфавита в поле $GF(2^{\omega})$ представляется многочленом длины ω вида

$$a(x) = \sum_{i=0}^{\omega-1} a_i x^i = a_{\omega-1}x^{\omega-1} + a_{\omega-2}x^{\omega-2} + \ldots + a_1 x + a_0,$$

где коэффициенты a_i являются бинарными, $a_i \in GF(2)$. Эти же коэффициенты составляют битовую запись символа a:

$$a = (a_{\omega-1}a_{\omega-2}\cdots a_1 a_0).$$

Дополнительно, символ a допускает блочное описание последовательностью из s машинных слов A_j разрядностью w ($\omega = sw$):

$$a = (A_{s-1}A_{s-2}\cdots A_1 A_0), A_j = \left(a_{j(w+1)-1}a_{j(w+1)-2}\cdots a_{jw+1}a_{jw}\right).$$

В этом случае, полиномиальные представления символа a и машинного слова A_j примут вид:

$$a(x) = \sum_{i=0}^{s-1} A_i(x)x^{iw} = A_{s-1}(x)x^{(s-1)w} + A_{s-2}(x)x^{(s-2)w} + \ldots + A_1(x)x^w + A_0(x),$$

$$A_j(x) = \sum_{i=0}^{w-1} a_{jw+i}x^i = a_{jw+(w-1)}x^{w-1} + a_{jw+(w-2)}x^{\omega-2} + \ldots + a_{jw+1}x + a_{jw}.$$

Если представить второй множитель символом b из того же алфавита:

$$b = (B_{s-1}B_{s-2}\cdots B_1 B_0),$$

то псевдокод алгоритма Монтгомери из работы [9] может быть представлен следующей программной реализацией на языке C++ (см. листинг 1).

Листинг 1. Функция перемножения длинных чисел в поле Галуа по алгоритму Монтгомери для $w = 32$.

```
union UIntGF2XT{
private:
    uint8_t u08[UIntGF2XTu08LEN];
    uint16_t u16[UIntGF2XTu16LEN];
    uint32_t u32[UIntGF2XTu32LEN];
    uint64_t u64[UIntGF2XTu64LEN];
public:
    friend const UIntGF2XT MULGF2(UIntGF2XT A, const UIntGF2XT& B);
    friend void GenAndInvPolyN32(UIntGF2XT& polyN,uint32_t& polyNinv);};
const (UIntGF2XT A, const UIntGF2XT& B){
    uint32_t i, j, H, L, P, M;
    uint32_t C[UIntGF2XTu32LEN],N[UIntGF2XTu32LEN],Caux,Ninv0;
    std::fill(C,C+UIntGF2XTu32LEN,'\0');
    Caux = 0ULL;
    GenAndInvPolyN32(&N,&Ninv0);
    for(i = 0; i < UIntGF2XTu32LEN; i++){
        for(j = 0; j < UIntGF2XTu32LEN - 1; j++){
            MULGF2_32(H, L, A.u32[j], B.u32[i]);
            C[j] ^= L;
            C[j+1] ^= H;}
        MULGF2_32(H, L, A.u32[j], B.u32[i]);
        C[j] ^= L;
        Caux ^= H;
        MULGF2_32(H, M, C[0], Ninv0);
        MULGF2_32(P, L, M, N[0]);
        for(j = 1; j < UIntGF2XTu32LEN; j++){
            MULGF2_32(H,L,M,N[j]);
            C[j-1] = C[j] ^ L ^ P;
            P = H;}
    C[j-1] = Caux ^ P ^ M;//j = UIntGF2XTu32LEN
    Caux = 0U;}
    std::copy(C,C+UIntGF2XTu32LEN,A.u32);
    return A;}
```

В этом листинге константа `UIntGF2XTu32LEN` соответствует количеству s машинных слов разрядностью $w = 32$. Массив `N[UIntGF2XTu32LEN]` представляет собой неприводимый многочлен

степени 2^ω в поле $GF(2)$, а машинное слово `Ninv0` является обратным многочленом к `N[0]`.

Как видно из листинга 1, в качестве основной алгебраической операцией используется функция умножения `MULGF2_32()`. Эта функция реализует улучшенный алгоритм Карацубы без рекурсии, представленный в работе [10] (см. листинг 2).

Листинг 2. Функция перемножения 32-разрядных чисел в поле Галуа по алгоритму Карацубы.

```
uint64_t MULGF2_32(uint32_t& H, uint32_t& L, uint32_t A, uint32_t B){
    uint16_t AH, AL, BH, BL;
    uint32_t Z0,Z1,Z2;
    uint64_t AB, ZZ;
    // A = AH*Bm + AL
    // B = BH*Bm + BL
    // Bm = 2^16
    // Z0 = AL*BL
    // Z1 = (AH-AL)*(BH-BL)
    // Z2 = AH*BH
    // AB = (Bm^2+Bm)*Z2 - Bm*Z1 + (Bm+1)*Z2
    AH = (uint16_t) (A >> 16);
    AL = (uint16_t) (A & 0x0000FFFF);
    BH = (uint16_t) (B >> 16);
    BL = (uint16_t) (B & 0x0000FFFF);
    Z0 = MULGF2_16(AL, BL);
    Z2 = MULGF2_16(AH, BH);
    if(AH >= AL){ AH -= AL; }
    else{ AH = AL - AH; }
    if(BH >= BL){ BH -= BL; }
    else{ BH = BL - BH; }
    Z1 = MULGF2_16(AH, BH);
    ZZ = ((uint64_t) Z2);
    AB = ( ZZ + ((uint64_t) Z0) ) - ((uint64_t) Z1);
    AB <<= 16;
    AB += ( ZZ << 32 ) + ((uint64_t) Z0);
    H = (uint32_t) (AB >> 32);
    L = (uint32_t) (AB & 0x00000000FFFFFFFF);
    return AB;}
```

Функция возвращает 64-разрядный результат перемножения двух 32-разрядных машинных слов, а также отдельные верхнее и нижнее 32-разрядные слова результата. Как видно из листинга, в программной реализации алгоритма Карацубы функцией `MULGF2_16()` перемножаются 16-разрядные машинные слова согласно следующим выражениям:

$$A = 2^{16}A_H + A_L,\ B = 2^{16}B_H + B_L,\ Z_2 = A_H B_H,\ Z_1 = (A_H - A_L)(B_H - B_L),\ Z_0 = A_L B_L,$$
$$AB = (2^{32} + 2^{16})Z_2 - 2^{16}Z_1 + (2^{16} + 1)Z_0.$$

Поскольку разрядность этих машинных слов невелика, в общем случае наиболее производительным решением является применение таблиц умножения [10], соответствующих правилу:

$$AB = \frac{(A^2 + B^2 - (A - B)^2)}{2}.$$

Подобные таблицы хранят только искомые квадраты значений величины и позволяют сократить используемый объем памяти с $2^\omega \times 2^\omega \times 2\omega$ бит до $2^\omega \times 2\omega$. Реализация функции перемножения 16-разрядных машинных слов приведена в листинге 3.

Листинг 3. Функция табличного перемножения 16-разрядных чисел в поле Галуа.

```
uint32_t LUT16[sz16x2];
void CreateLUT16(){
    uint32_t i;
    LUT16[0] = (uint32_t)0;
    for(i = 0; i < sz16x2-1; i++){
        LUT16[i+1] = LUT16[i] + 2*i + 1;}
}
inline uint32_t MULGF2_16(uint16_t A, uint16_t B)
{
    uint32_t AsubB, Asqr, Bsqr, AsubBsqr;
    uint32_t AB;
    Asqr = LUT16[A];
    Bsqr = LUT16[B];
    if(A>=B)
    {
        AsubB = A - B;
```

```
        AsubBsqr = LUT16[AsubB];
        AB = (((Asqr - AsubBsqr) >> 1) + ((Bsqr + 1) >> 1));
    }
    else
    {
        AsubB = B - A;
        AsubBsqr = LUT16[AsubB];
        AB = (((Bsqr - AsubBsqr) >> 1) + ((Asqr +1)  >> 1));
    }
    return AB;
}
```

Таким образом, в работе приведена оригинальная программная реализация наиболее вычислительно ёмких алгоритмов, используемых в методе кодирования Рида—Соломона. Показано, что можно найти компромиссный вариант алгоритма умножения в поле Галуа, использующий как таблицы для операндов малой разрядности, так и методы умножения Карацубы для операндов высокой разрядности. Подобный подход к вычислению произведений множителей позволяет построить коды Рида—Соломона в конечных полях с алфавитами большой мощности, подходящие под требования для создания систем хранения данных, устойчивых к сбоям и ошибкам в накопителях.

Работа проведена в рамках выполнения прикладных научных исследований при финансовой поддержке Министерства образования и науки Российской Федерации. Соглашения о предоставлении субсидий № 14.579.21.0010. Уникальный идентификатор Соглашения RFMEFI57914X0010.

Литература

1. Ivanichkina L. Computer Simulator of Failures in Super Large Data Storage / L. Ivanichkina, A. Neporada // Contemporary Engineering Sciences. – 2015. – Т. 8. – № 28. – С. 1679–1691.
2. Ivanichkina L. Mathematical methods and models of improving data storage reliability including those based on finite field theory / L. Ivanichkina, A. Neporada // Contemporary Engineering Sciences. – Т. 7. –№ 28. – 2014. – С. 1589 – 1602.
3. ETSI EN 300 744 V1.6.1 (2009-01) Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television // European Standard (Telecommunications series). – 2009. – 66 С.
4. Seroussi G. Table of low-weight binary irreducible polynomials. – Hewlett-Packard Laboratories, 1998.
5. Карацуба А. А., Офман Ю. П. Умножение многозначных чисел на автоматах //ДАН СССР. – 1961. – Т. 145. – № 2. – С. 293-294.
6. Toom A. L. The complexity of a scheme of functional elements realizing the multiplication of integers //Soviet Mathematics Doklady. – 1963. – Т. 3. – №. 4. – С. 714-716.
7. Schönhage D. D. A., Strassen V. Schnelle multiplikation grosser zahlen //Computing. – 1971. – Т. 7. – № 3-4. – С. 281-292.
8. Fürer M. Faster integer multiplication //SIAM Journal on Computing. – 2009. – Т. 39. – № 3. – С. 979-1005.
9. Koc C. K., Acar T. Montgomery multiplication in GF (2k) //Designs, Codes and Cryptography. – 1998. – Т. 14. – № 1. – С. 57-69.
10. Wang B. F., Chen C. L., Chen G. H. A simple approach to implementing multiplication with small tables //Information Processing Letters. – 1991. – Т. 37. – № 6. – С. 327-329.

References

1. Ivanichkina, L., and Neporada, A. (2015). Computer Simulator of Failures in Super Large Data Storage. Contemporary Engineering Sciences.–2015.–8 (28).–С, 1679-1691.
2. Ivanichkina, L., and Neporada, A. (2014). Mathematical methods and models of improving data storage reliability including those based on finite field theory. Contemporary Engineering Sciences, 7(28), 1589-1602.
3. ETSI EN 300 744 V1.6.1 (2009-01) Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television // European Standard (Telecommunications series), 66 p.
4. Seroussi, G. (1998). Table of low-weight binary irreducible polynomials. Hewlett-Packard Laboratories.
5. Karacuba, A. A. (1975). Berechnungen und die Kompliziertheit von Beziehungen. Elektronische Informationsverarbeitung Kybernetik, 11, 603-606.
6. Toom, A. L. (1963, June). The complexity of a scheme of functional elements realizing the multiplication of integers. In Soviet Mathematics Doklady (Vol. 3, No. 4, pp. 714-716).
7. Schönhage, D. D. A., and Strassen, V. (1971). Schnelle multiplikation grosser zahlen. Computing, 7(3-4), 281-292.
8. Fürer, M. (2009). Faster integer multiplication. SIAM Journal on Computing, 39(3), 979-1005.
9. Koc, C. K., and Acar, T. (1998). Montgomery multiplication in GF (2k). Designs, Codes and Cryptography, 14(1), 57-69.
10. Wang, B. F., Chen, C. L., and Chen, G. H. (1991). A simple approach to implementing multiplication with small tables. Information Processing Letters, 37(6), 327-329.

Поступила: 14.10.2016

Об авторах:

Винников Владимир Владимирович, технический писатель ООО «Акронис», старший научный сотрудник отдела вычислительной физики Вычислительного центра им. А.А. Дородницына Федерального исследовательского центра «Информатика и управление» Российской академии наук, кандидат физико-математических наук, vvinnikov@list.ru;

Иваничкина Людмила Владимировна, аспирант факультета управления и прикладной математики Московского физико-технического института, старший разработчик ООО «Проект Икс».