УДК 004.056.3+004.056.4

В. М. Пименов [1], **Е. В. Соколов** [2], **А. Л. Кобец** [3]

[1] Институт автоматизации проектирования РАН
ул. 2-я Брестская, 19/18, Москва, 123056, Россия
E-mail: __vm@mail.ru

[2, 3] Московский физико-технический институт (государственный университет)
пер. Институтский, 9, Долгопрудный, Московская обл., 141700, Россия
E-mail: [2] kason@yandex.ru, [3] kobets@sw.ru

СПОСОБЫ УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ АЛГОРИТМОВ ДЛЯ ОТКАЗОУСТОЙЧИВЫХ СИСТЕМ ХРАНЕНИЯ ДАННЫХ

Современные информационные системы обрабатывают все возрастающие объемы данных, которые требуют как места для хранения, так и высокой пропускной способности для передачи по сети. Для обеспечения сохранности данных применяются различные способы, основанные в первую очередь на хранении с избыточностью (как правило, путем *n*-кратного копирования или с использованием заранее определенных схем создания избыточности [Patterson, 1988]).

Ранее был предложен подход [Тормасов и др., 2001; Пименов, Сметанин, 2006], основанный на (n,k)-пороговой схеме, позволяющий более гибко управлять избыточностью хранения данных. Именно, данные преобразовываются в n частей («разборка»), по любым k которых можно восстановить исходные данные («сборка»). Это дает возможность выбирать соотношение между надежностью и используемым для хранения объемом. Алгоритм основывается на использовании свойств матрицы Вандермонда, любые k из n строк которой являются линейно независимыми. Матрица Вандермонда размером $n \times k$ строится из n различных порождающих элементов p_i, последовательно возводимых в степень: $a_{ij} = p_i^{j-1}$, $i \in [1;n]$, $j \in [1;k]$. Исходный файл разбивается на последовательность ячеек x_i, собираемых в матрицу, и умножается слева на матрицу Вандермонда:

$$\begin{pmatrix} y_1 & y_{n+1} & y_{2n+1} & \cdots \\ y_2 & y_{n+2} & y_{2n+2} & \cdots \\ \vdots & \vdots & \vdots & \cdots \\ y_{n-1} & y_{2n-1} & y_{3n-1} & \cdots \\ y_n & y_{2n} & y_{3n} & \cdots \end{pmatrix} = \begin{pmatrix} 1 & p_1 & \cdots & p_1^{k-1} \\ 1 & p_2 & \cdots & p_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & p_{n-1} & \cdots & p_{n-1}^{k-1} \\ 1 & p_n & \cdots & p_n^{k-1} \end{pmatrix} \begin{pmatrix} x_1 & x_{k+1} & x_{2k+1} & \cdots \\ x_2 & x_{k+2} & x_{2k+2} & \cdots \\ \vdots & \vdots & \vdots & \cdots \\ x_k & x_{2k} & x_{3k} & \cdots \end{pmatrix}.$$

Каждая полученная строка (проекция) вместе с соответствующим ей порождающим элементом сохраняется в отдельном месте.

Для выполнения обратного преобразования требуются любые k проекций вместе со значениями p_i, использовавшимися при их построении. Из подмножества p_i по правилу построения исходной матрицы $\|M\|$ составляется матрица $\|\tilde{M}\|$. Так как построенная матрица тоже является матрицей Вандермонда, то все ее строки линейно независимы. Поэтому можно вычислить обратную к ней матрицу $\|\tilde{M}\|^{-1}$. Тогда исходные векторы $\vec{x_j}$ получаются по формуле $\vec{x_j} = \|\tilde{M}\|^{-1} \vec{y_j}$:

$$\begin{pmatrix} x_1 & x_{k+1} & x_{2k+1} & \cdots \\ x_2 & x_{k+2} & x_{2k+2} & \cdots \\ \vdots & \vdots & \vdots & \cdots \\ x_k & x_{2k} & x_{3k} & \cdots \end{pmatrix} = \begin{pmatrix} 1 & p_{a_1} & \cdots & p_{a_1}^{k-1} \\ 1 & p_{a_2} & \cdots & p_{a_2}^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & p_{ak} & \cdots & p_{ak}^{k-1} \end{pmatrix}^{-1} \begin{pmatrix} y_{a_1} & y_{n+a_1} & y_{2n+a_1} & \cdots \\ y_{a_2} & y_{n+a_2} & y_{2n+a_2} & \cdots \\ \vdots & \vdots & \vdots & \cdots \\ y_{a_k} & y_{n+a_k} & y_{2n+a_k} & \cdots \end{pmatrix}.$$

ISSN 1818-7900. Вестник НГУ. Серия: Информационные технологии. 2007. Том 5, выпуск 1

Скорость разборки с использованием данного алгоритма обратно пропорциональна nk, а сборки – k^2, и, зная скорость разборки, можно получить скорость сборки, умножив первую на n/k. Поэтому свойства алгоритмов сборки в работе практически не рассматриваются (за исключением особо оговоренных случаев).

Работа (n,k)-схемы обеспечивается свойствами матрицы Вандермонда, но ячейки не могут представляться обычными рациональными числами в силу того, что их множество неограничено, а в вычислительной технике применяются конечные числа. Поэтому в качестве ячеек матриц используются элементы полей Галуа, вычисления в которых достаточно трудоемки. В работе рассматривается поле $GF(2^8)$, представимое в виде многочленов 7 степени с коэффициентами из поля $GF(2)$. Элементы этого поля можно описывать [1] байтами, где каждый разряд задает коэффициент при соответствующей степени. В таком случае сложение выполняется операцией «исключающее ИЛИ», а умножение – по правилам перемножения многочленов по модулю какого-либо неприводимого многочлена 8 степени.

В существующих процессорах общего назначения отсутствуют команды, выполняющие умножение в полях Галуа. Программное вычисление «в лоб» по правилам перемножения многочленов чрезвычайно медленно. Общепринятым способом преодоления этого ограничения для $GF(2^8)$ является использование таблиц умножения и деления, содержащих заранее вычисленные значения соответствующих операций [2]. Однако скорость работы подобных алгоритмов также оставляет желать лучшего (рис. 1).

В настоящей работе описываются различные способы увеличения скорости преобразований по (n,k)-схеме на процессорах общего назначения с архитектурами intel / amd x86 / x64 как за счет использования низкоуровневых свойств процессоров, так и за счет изменения свойств алгоритмов, а также благодаря ускорению вычислений в полях Галуа.

Разворачивание циклов

При выполнении конкретного преобразования (когда матрица преобразования $\|M\|$ уже зафиксирована) можно использовать локальную таблицу умножения, полученную копированием из общей таблицы строк, соответствующих задействованным в $\|M\|$ элементам, в один массив друг за другом. Это позволяет расположить участки с большой частотой обращения рядом друг с другом (в отличие от полной таблицы, где используемые и неиспользуемые строки перемешаны). Это приводит к более эффективному использованию кешей процессора, а также сокращает число используемых ячеек таблицы умножения. Кроме того, так как число столбцов матрицы $\|M\|$ уже известно, можно осуществить «разворачивание» внутреннего цикла (loop unrolling) [3], проходящего по каждой из ее строк, следующим образом. Для получения i-й проекции используется только одна строка матрицы преобразования $(1\ p_i\ \ldots\ p_i^{k-1}) \to (0\ 1\ \ldots\ k-1)$. Из таблицы умножения в одномерный массив выбираются лишь задействованные в преобразовании строки:

$$\begin{pmatrix} m_1^1 & m_1^2 & \ldots & m_1^{256} \\ \ldots & \ldots & \ldots & \ldots \\ m_{p_i}^1 & m_{p_i}^2 & \ldots & m_{p_i}^{256} \\ \ldots & \ldots & \ldots & \ldots \\ m_{p_i^{k-1}}^1 & m_{p_i^{k-1}}^2 & \ldots & m_{p_i^{k-1}}^{256} \end{pmatrix}.$$

Значения ячеек заменяются индексами строк массива: $(1\ p_i\ \ldots\ p_i^{k-1}) \to (0\ 1\ \ldots\ k-1)$, и вместо цикла из k шагов само тело цикла записывается k раз, с использованием явных смещений.

Результаты применения подхода «разворачивание циклов» приведены на рис. 1 в сравнении с исходной реализацией (n,k)-схемы.

---

[1] Более подробное и строгое описание можно найти в [Пименов, Сметанин, 2006].

[2] Использование «таблицы умножения» дает 22-кратный прирост скорости разборки на x86 архитектуре.

[3] Intel architecture software developer's manual. Vol. 3: system programming guide, 1999. http://www.intel.com/design/pentiumii/manuals/243192.htm.

Упрощение матрицы преобразования

«Алгоритмическим» способом ускорения является предварительное упрощение матрицы Вандермонда, которое заключается в сведении элементарными преобразованиями матрицы $\|M\|$ к матрице $\|\widetilde{M}\|$ вида $\left\|\frac{E}{K}\right\|$, где $\|E\|$ – единичная матрица:

$$\|\widetilde{M}\| = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ a_{k+1,1} & a_{k+1,2} & \dots & a_{k+1,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,k} \end{pmatrix}.$$

Полученная матрица $\|\widetilde{M}\|$ обладает теми же свойствами, необходимыми для реализации (n,k)-схемы, что и исходная $\|M\|$ (любые n из k строк являются линейно независимыми и могут образовать базис в k-мерном пространстве).

В исходном алгоритме для вычисления $\vec{Y} = \|M\|\vec{X}$ требовалось kn операций умножения. После преобразования матрицы $\|M\|$ можно представить вычисления $\overrightarrow{Y\uparrow} = \|E\|\vec{X}\|Y_{\uparrow}\| = \|E\|\vec{X}$ и $\overrightarrow{Y\downarrow} = \|K\|\vec{X}$. Так как преобразованные с помощью единичной матрицы данные совпадают

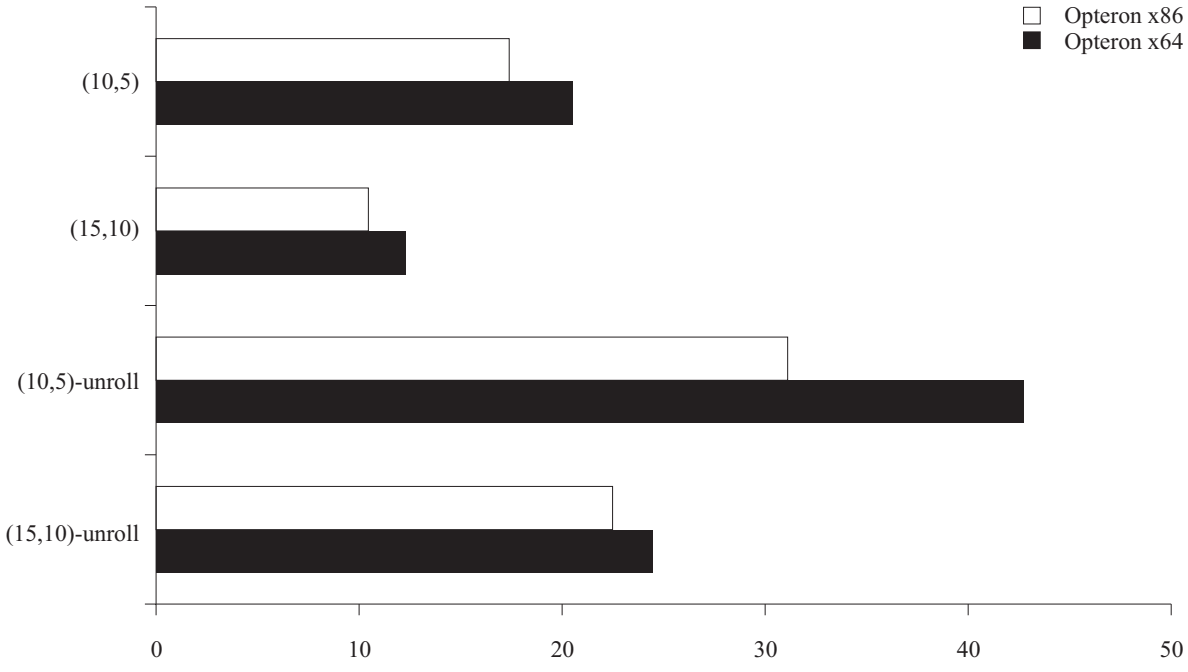


Рис. 1. Скорости работы разборки файлов в исходной и ускоренной реализациях, МБ/с

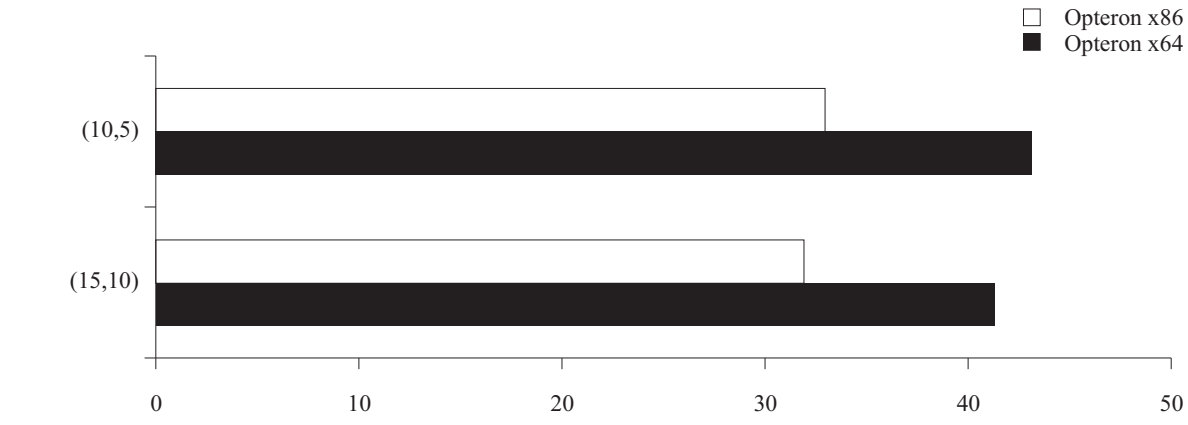


Рис. 2. Скорости работы разборки файлов при использовании «упрощенной» матрицы преобразования, МБ/с

с исходными, остается вычислить только нижнюю часть вектора $\overrightarrow{Y\downarrow}$, и поэтому количество требуемых операций умножения уменьшается до $k(n-k)$.

Умножение на единичную матрицу фактически представляет собой копирование, но в силу выбранного способа размещения исходных данных и проекций, сопровождается переупорядочением. Для того чтобы избежать вызываемого этим замедления, порядок следования исходных данных изменяется (рис. 2):

$$\begin{pmatrix} y_1 & y_2 & y_3 & \dots \\ y_{m+1} & y_{m+2} & y_{m+3} & \dots \\ \vdots & \vdots & \vdots & \dots \\ y_{m(n-1)+1} & y_{m(n-1)+2} & y_{m(n-1)+3} & \dots \\ y_{mn+1} & y_{mn+2} & y_{mn+3} & \dots \end{pmatrix} = \begin{pmatrix} 1 & p_1 & \dots & p_1^{k-1} \\ 1 & p_2 & \dots & p_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & p_{n-1} & \dots & p_{n-1}^{k-1} \\ 1 & p_n & \dots & p_n^{k-1} \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 & \dots \\ x_{m+1} & x_{m+2} & x_{m+3} & \dots \\ \vdots & \vdots & \vdots & \dots \\ x_{mk+1} & x_{mk+2} & x_{mk+3} & \dots \end{pmatrix}.$$

Скорость сборки с использованием «упрощенной» матрицы зависит от выбираемых проекций. Это происходит из-за того, что число строк, выписанных из единичной матрицы, в $\|\tilde{M}\|$ и $\|\tilde{M}\|^{-1}$ совпадает, поэтому данные либо просто копируются, либо преобразуются с использованием операций в полях Галуа (рис. 3).

Матрица преобразования с единичным столбцом [4]

Свойство независимости строк матрицы сохраняется, если любую строку матрицы разделить на произвольный коэффициент. Поэтому ее можно упростить, преобразовав все коэффициенты нижней части по формуле

$$b_{ij} = a_{ij}/a_{ij}, \quad k+1 \le n, \quad 1 \le j \le k.$$

Таким образом матрица принимает вид

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ 1 & b_{k+1,2} & \dots & b_{k+1,k} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_{n,2} & \dots & b_{n,k} \end{pmatrix},$$

а количество умножений уменьшается на $n-k$ (рис. 4).

Вычисления в полях $GF(2^4)$

Для обеспечения отказоустойчивости в локальных системах хранения данных нередко используются RAID-массивы, состоящие из нескольких дисков. В подобных системах возможно использование (n,k)-схемы на основе $GF(2^4)$ [5], при этом порядок n и k (ограничен-

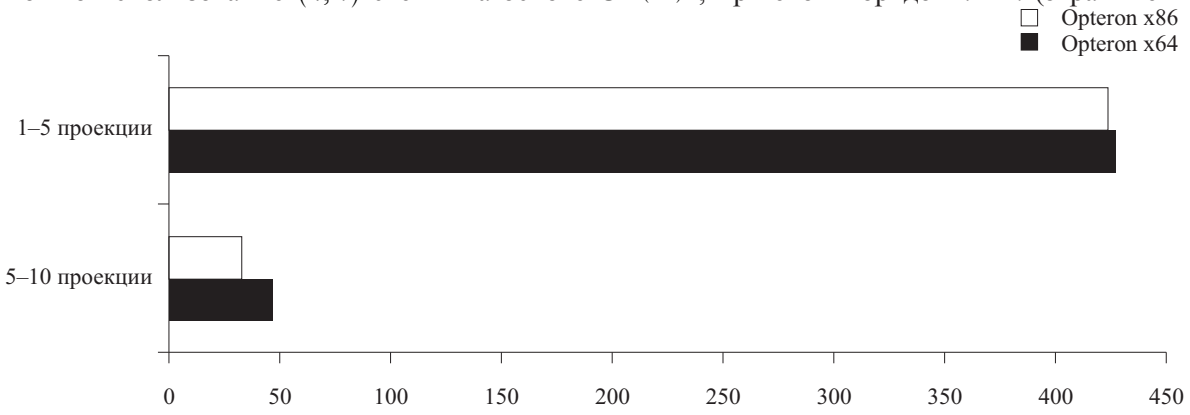


Рис. 3. Скорости работы сборки файлов при использовании «упрощенной» матрицы преобразования при $(n,k) = (10,5)$, МБ/с

[4] Этот алгоритм ускорения разборки не может быть применен для сборки.

[5] Более подробное описание выполнения операций в этих полях можно найти в работах [Mastrovito, 1991; Paar, 1994]. См. также: Advanced encryption standard (AES) / National Institute of Standards and Technology (NIST), 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

ных 15) вполне достаточен для работы. В этом случае операция умножения становится вычислимой напрямую через последовательность «простых» арифметических операций (имеющихся и в SIMD-командах), и необходимость использования таблицы умножения отпадает. Эти вычисления осуществляются следующим образом.

Элементы поля $GF(2^4)$ можно представить в виде многочленов 3-й степени:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0,$$
$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0.$$

Результат их умножения:

$$c(x) = a(x)b(x) \bmod p(x),$$

где $p(x)$ – любой неприводимый многочлен 4-й степени, например, $x^4 + x + 1$. В таком случае

$$c(x) = c_3x^3 + c_2x^2 + c_1x + c_0,$$
$$c_0 = (a_0 \otimes b_0) \oplus \underbrace{(a_3 \otimes b_1) \oplus (a_2 \otimes b_2) \oplus (a_1 \otimes b_3)}_{\tilde{c}_0},$$
$$c_1 = (a_1 \otimes b_0) \oplus (a_0 \otimes b_1) \oplus \underbrace{(a_3 \otimes b_2) \oplus (a_2 \otimes b_3)}_{\tilde{c}_1} \oplus \tilde{c}_0,$$
$$c_2 = (a_2 \otimes b_0) \oplus (a_1 \otimes b_1) \oplus (a_0 \otimes b_2) \oplus \underbrace{(a_3 \otimes b_3)}_{\tilde{c}_2} \oplus \tilde{c}_1,$$
$$c_3 = (a_3 \otimes b_0) \oplus (a_2 \otimes b_1) \oplus (a_1 \otimes b_2) \oplus (a_0 \otimes b_3) \oplus \tilde{c}_2,$$

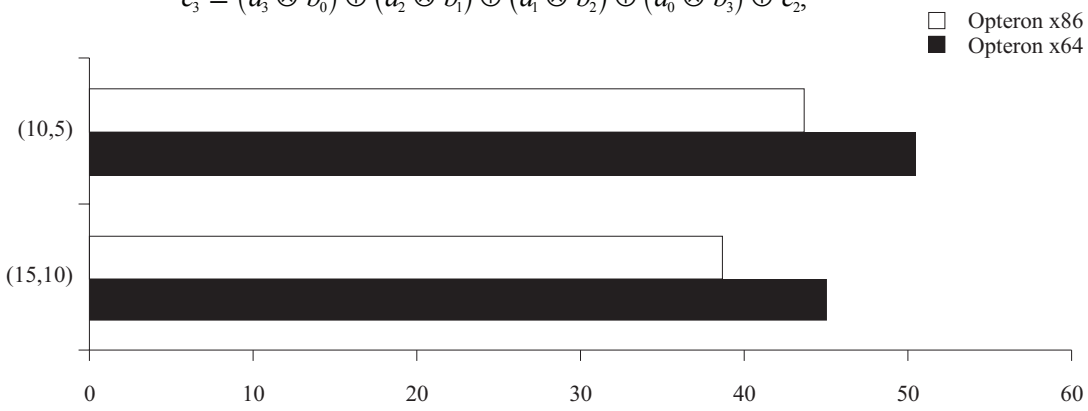


Рис. 4. Скорости работы разборки файлов при использовании матрицы преобразования с единичным столбцом, МБ/с

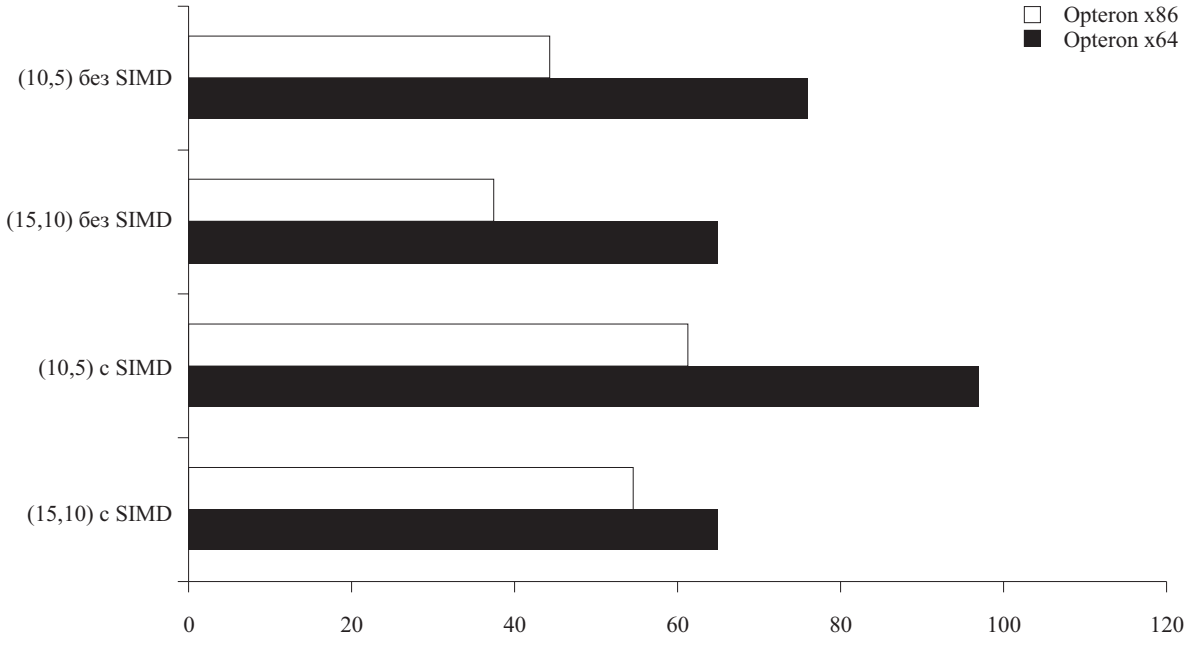


Рис. 5. Скорости работы разборки файлов при использовании $GF(2^4)$ совместно с «упрощенной» матрицей на обычных и SSE-регистрах, МБ/с

где сгруппированы одинаковые слагаемые. При неизменном $b(x)$ можно представить результат в матричной форме:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \underset{\sim}{c_3} \\ c_0 \\ \vdots \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ \underset{\sim}{a_3} & \underset{\sim}{a_2} & \underset{\sim}{a_1} & \underset{\sim}{a_0} \\ a_0 & a_3 & a_2 & a_1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \otimes \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ (a_3 \otimes b_1) \oplus (a_2 \otimes b_2) \oplus (a_1 \otimes b_3) \\ (a_3 \otimes b_2) \oplus (a_2 \otimes b_3) \\ (a_3 \otimes b_3) \\ \hline 0 \\ \vdots \end{pmatrix}.$$

После этого каждый столбец матрицы $\|a_{ij}\|$ помещается в отдельный регистр и целиком умножается на одну ячейку b_j. Часть произведения используется для вычисления правого столбца (с использованием циклического сдвига и маскирования «логическим и»), складывается с ним и добавляется к предыдущему результату. После повторения этих действий для всех столбцов матрицы $\|a_{ij}\|$ получается столбец $\|c_j\|$ с результатом.

Алгоритм реализован с использованием двойных слов (64 разряда) и SSE-регистров и команд (128 разрядов). Результаты работы алгоритма приведены на рис. 5.

Параллельное выполнение

Преобразование по (n,k)-схеме линейно. Поэтому можно разделить всю работу на части и выполнять их параллельно на нескольких процессорах. На рис. 6 приведена производительность алгоритма в зависимости от количества одновременно выполняемых потоков на двухпроцессорной машине.

Выводы

Общий график производительности разборки (МБ/с) разных алгоритмов представлен на рис. 7. Отмечен порядок следования различных алгоритмов и величина относительного прироста скорости. Первоначально независимо были реализованы «разворачивание циклов» и «упрощение матрицы». Затем на основе «упрощенной матрицы» реализована «матрица с единичным столбцом». На основе последнего сделаны обе реализации $GF(2^4)$. «Распараллеливание» реализовано на основе «$GF(2^4)$ без SIMD».

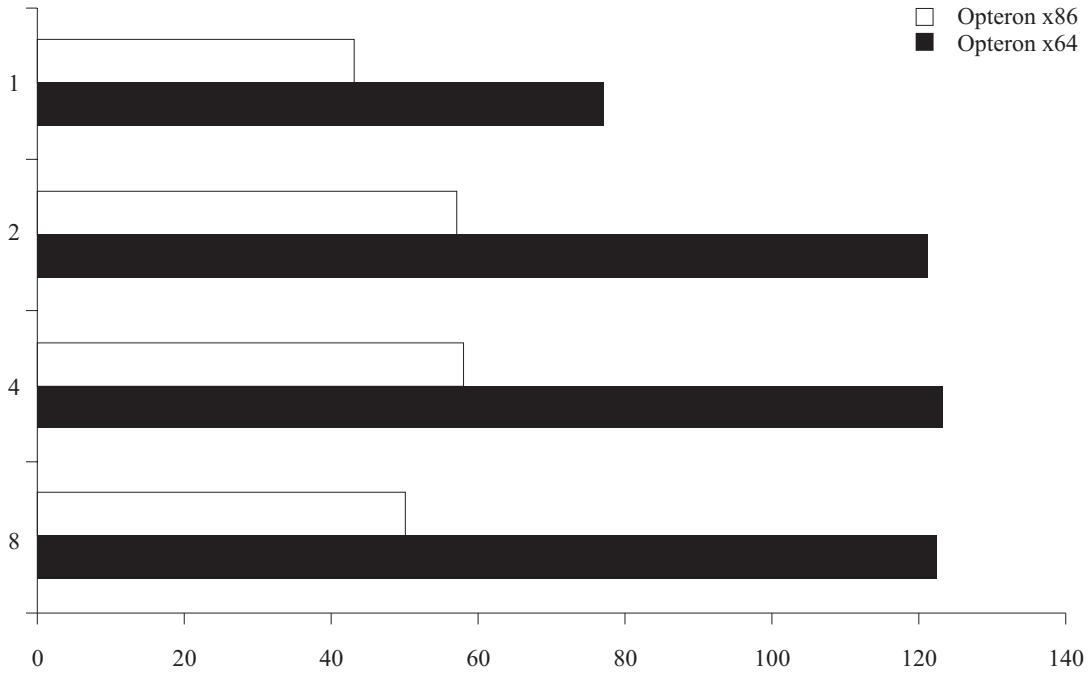
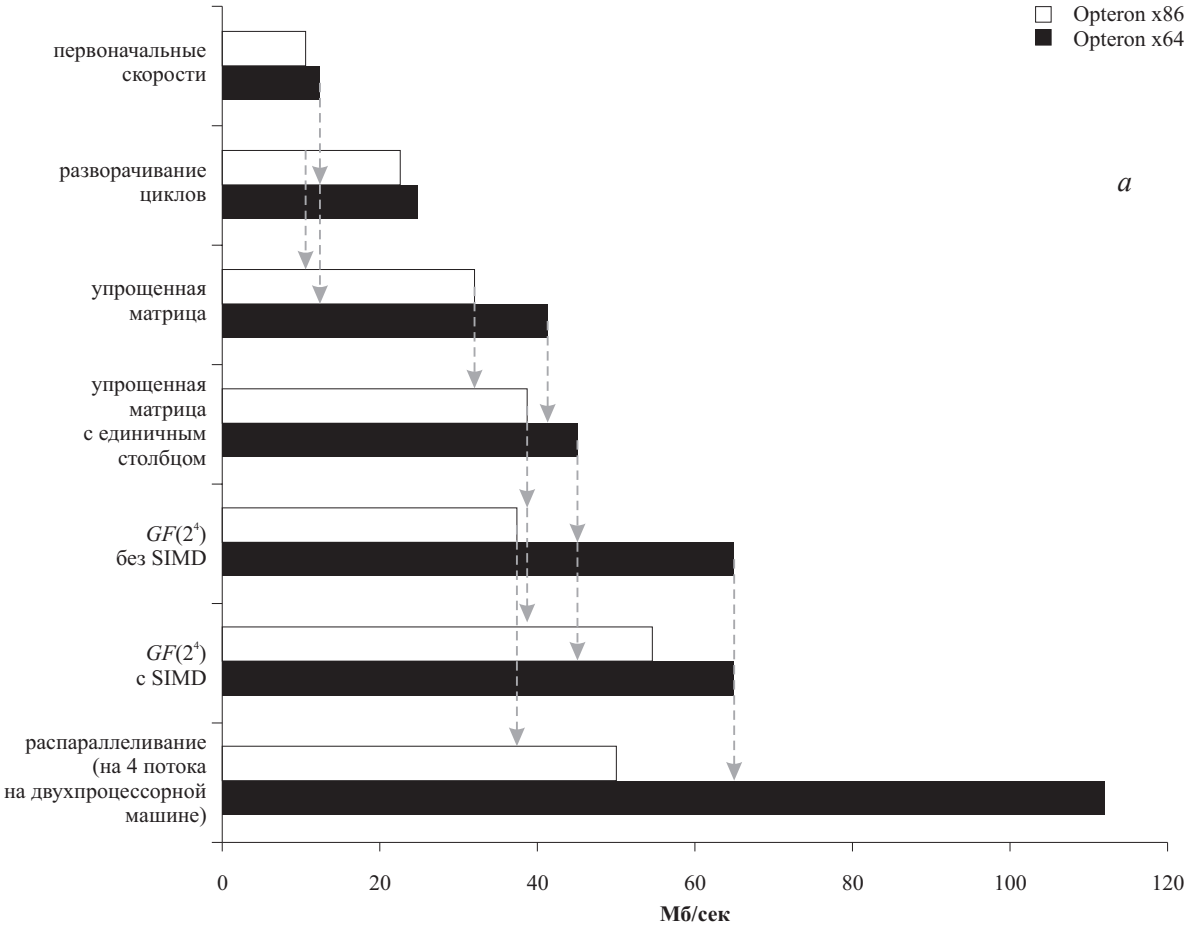


Рис. 6. Скорости работы разборки файлов при параллельном выполнении вычислений в $GF(2^4)$ совместно с «упрощенной» матрицей на двухпроцессорной машине при $(n,k) = (10,5)$ в зависимости от количества одновременно работающих потоков, МБ/с



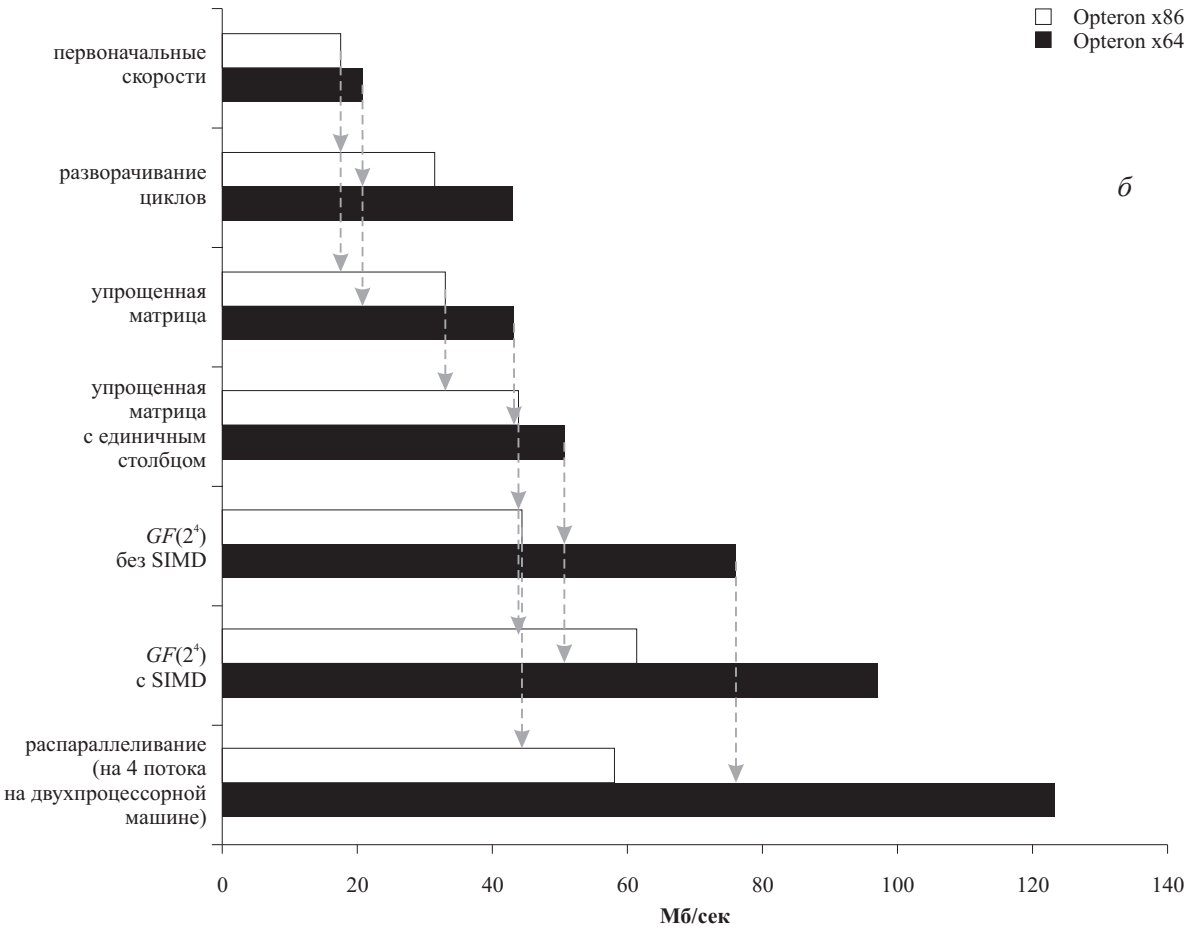


Рис. 7. Общий график производительности алгоритмов: *а* – для $(n,k)=(10,5)$; *б* – для $(n,k)=(15,10)$.

– без распараллеливания достигнуто ускорение от 3-х до 5-ти раз в зависимости от выбора n и k;

– распараллеливание дает дополнительное ускорение в 1,6 раз на двухпроцессорной машине;

– за счет удачного подбора проекций при сборке можно достигнуть большей скорости; наименее ресурсоемкими являются первые k проекций;

– достигнутые скорости позволяют эффективно использовать (n,k)-схему в гигабитных сетях для распределенного хранения данных.

Список литературы

Пименов В. М., Сметанин А. Г. Использование программируемых графических процессоров в задачах хранения данных // Проблемы вычислительной математики, математического моделирования и информатики. М.: МЗ Пресс, 2006. С. 138–157.

Тормасов А. Г., Хасин М. А., Пахомов Ю. И. Модель распределенного хранения данных с регулируемой избыточностью // Электронный журнал «Исследовано в России». Т. 4. 2001. С. 355–364. http://zhurnal.ape.relarn.ru/articles/2001/035.pdf.

Mastrovito E. D. VLSI Architectures for Computation in Galois Fields. PhD thesis. Linköping Univ., Sweden, 1991.

Patterson D., Gibson G., Katz R. A Case for Redundant Arrays of Inexpensive Disks (RAID) // Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data. Chicago: 1988. P. 109–116.

Paar C. Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields. PhD thesis (English translation). Inst. for Experimental Math., Univ. of Essen. Essen, 1994.

Материал поступил в редколлегию 20.04.2007