# Многопоточные сервера, использующие обработчики событий

**Александров Д. Е.** (Москва, МГУ им. М. В. Ломоносова)

*d06alexandrov@gmail.com*

Исследуются основные архитектуры TCP серверов, реализуемых на SMP-системах. Выделены параметры архитектурных решений, характеризующие производительность серверов. На основании полученных данных разработан сервер с архитектурой, превосходящей по производительности существующие архитектуры.

## Введение

Один из основных вопросов при выборе сервера для реализации какого-либо протокола — баланс между производительностью обработки запросов и низкими затратами при портировании готовых программных решений на базу интернет-сервера. Использование обработчиков событий позволяет существенно сократить расходы на обработку группы соединений, что позволяет эффективно использовать кэширование часто используемых данных.

Для анализа интернет-серверов на базе SMP-систем были выделены основные параметры, позволяющие сделать сравнительные оценки производительности архитектур. На основании полученных данных была спроектирована гибридная архитектура, превосходящая по этим характеристикам существующие архитектуры.

## Серверные архитектуры

Основными серверными архитектурами являются:

- Однопоточная событийно-зависимая. *Squid proxy* [4].
- Многопроцессная. *Apache web-server (MPM worker module)* [5]
- Многопоточная. *Apache web-server* [5]
- Многопоточная событийно-зависимая. *Flash web-server* [2]

При изучении архитектур были выделены следующие параметры, влияющие на производительность:

- Количество одновременно обрабатываемых соединений.
- Переключение контекста.
- Адресное пространство.
- Портируемость кода.

**Гибридная архитектура**

Основой этой архитектуры является главный поток, синхронизирующий деятельность рабочих потоков. Рабочие потоки — потоки, осуществляющие непосредственную обработку соединения путем переключения контекста с потока на обработчик.

В самом начале работы сервера создается набор пустых обработчиков соединений (одно соединение — один обработчик), являющихся облегченными версиями стандартных потоков. Каждый обработчик имеет свой стек вызовов и хранит основную информацию о соединении. Соединению, создаваемому при подключении к серверу, ставится в соответствие свободный обработчик. Далее главный поток проверяет, есть ли свободные рабочие потоки или нет. Если нет, то обработчик ставится в очередь. В противном случае к одному из свободных рабочих потоков отправляется соответствующий запрос на исполнение кода обработчика. Когда запрос с информацией об обработчике приходит, рабочий поток меняет свой контекст выполнения на контекст обработчика и начинает (продолжает) выполнение кода обработки соединения. Когда требуется получить данные с неблокирующего сокета, соответствующее сообщение отправляется главному потоку, и рабочий поток ожидает следующей команды, выходя из контекста текущего обработчика, а главный поток добавляет в свой список ожидаемых событий событие, затребованное обработчиком.

**Результаты общего сравнения**

По результатам теоретического сравнения основных параметров различных архитектур можно сделать вывод о преимуществе гибридной архитектуры. Во-первых, количество переключений контекстов потоков сокращается до минимума по сравнению с многопроцессными и многопоточными серверами. Кроме того, все рабочие потоки

действуют в рамках одного адресного пространства, позволяя организовывать различные механизмы кэширования. Во-вторых, имеется возможность портировать готовый исходный код под гибридную архитектуру без больших затрат, в отличие от событийно-зависимых архитектур.

### Особенности реализации

Количество одновременно обрабатываемых соединений физически не может превышать количество доступных ядер процессоров. В случае, если задать количество рабочих потоков приблизительно равным количеству ядер процессоров, мы получаем максимум количества параллельно обрабатываемых соединений.

Важнейшей частью гибридной архитектуры является реализация своего переключения контекста. Так как на высоконагруженных серверах количество ядер/процессоров всегда будет на несколько порядков меньше, чем количество соединений, то необходимо максимально снизить затраты на смену контекста. В стандартной реализации потоков в `Linux` помимо непосредственно переключения контекста (сохранение/загрузка некоторых системных регистров, в том числе регистров стека вызовов), производится довольно большое количество дополнительных действий. Нам же достаточно реализовать функции смены контекста, при использовании которых будут восстанавливаться стек и часть регистров согласно `AMD64 ABI` [1, ch.3.2.1] (стандарт реализации функций 64х разрядных систем). В рамках такого подхода были реализованы несколько основных функций: `code_start()` — инициализация стека обработчика и запуск функции обработки соединения, `code_pause()` — приостановка выполнения кода обработчика с возможностью продолжить при вызове команды `switch_context()`.

Благодаря тому, что у нас имеется только одно адресное пространство и затраты на процесс «общения» между рабочими и главным потоками малы, мы имеем возможность кэшировать часто запрашиваемые данные. Главный поток хранит информацию об открытых файлах, которые были отображены на память процесса с помощью системного вызова `mmap()`. Когда рабочему потоку требуется открыть

или прочитать информацию из файла, он «спрашивает» у главного потока, открыт ли данный файл. Если данный файл открыт и отображен на память, то чтение производится из кэша. В противном случае самостоятельно открывает файл и в зависимости от настроек сервера и объема свободной оперативной памяти может сам осуществить кэширование нового файла.

### Тестирование

Для проведения тестов был реализован простой веб-сервер (разбор простого `GET`-запроса и отправка файла в качестве ответа). Благодаря портируемости данный код был использован и на сервере с гибридной архитектурой, и на сервере с многопоточной архитектурой.

Тестирование показало, что количество запущенных потоков действительно играет важную роль в производительности сервера. Сервер с гибридной архитектурой (4 потока) имел реальную скорость передачи данных — около 70МБ/с (погрешность вычислений не больше 1%). Многопоточный же сервер (200 потоков) имел скорость около 65МБ/с. Таким образом разница составила около $5-8\%$. Кроме того исследования показали, что увеличение рабочих потоков гибридной системы (при неизменности доступных ядер процессора) приводит к снижению производительности.

### Заключение

Результатом исследования архитектур, используемых на данный момент, стала предложенная гибридная архитектура, сочетающая в себе преимущества событийно-зависимых серверов и многопоточных серверов. Когда объем свободной памяти допускает — используются механизмы первого типа серверов. При большой нагрузке — работа сервера схожа с работой многопоточных серверов, однако превосходит по производительности, так как минимизировано громоздкое переключение контекста. Кроме того, благодаря низкоуровневому программированию, стало возможным портировать уже реализованные алгоритмы на данный сервер без дополнительных трудозатрат.

Выдвинутые теоретические предположения были подтверждены тестированием разработанного сервера.

Автор выражает благодарность своему научному руководителю Панкратьеву Антону Евгеньевичу и Галатенко Алексею Владимировичу за общее руководство и помощь в проведении тестирования разработанного сервера.

**Список литературы**

[1] Matz M., Hubicka J., Jaeger A., Mitchell M. System V Application Binary Interface AMD64 Architecture Processor Supplement 0.99.5. — September 3, 2010.

[2] Zeldovich N. Concurrency Control for Multi-Processor Event-Driven Systems. — Massachusetts Institute of Technology, 2002.

[3] Bovet D. P., Cesati M. Understanding the Linux Kernel. — O'Reilly Media, 2000.

[4] http://www.squid-cache.org.

[5] http://httpd.apache.org.

# Методы оптимизации глубины реализации хэш-функций

**Болотов А. А., Галатенко А. В., Гринчук М. И., Золотых А. А., Иванович Л.**

*agalat@msu.ru*

## Введение

В работе предлагается ряд методов оптимизации схемной реализации арифметических алгоритмов хэширования: семейства SHA-2 [1] и функций SHA-1 [1] и MD5 [2]. Рассматриваемые алгоритмы хэширования состоят из двух этапов — предобработки и собственно выработка хэша. Предобработка включает в себя выравнивание сообщения, разрезание на блоки и инициализацию значений. На этом этапе трудоемких вычислений не производится. Выработка хэша заключается в итеративной обработке блоков сообщения; именно эта часть представляет основную сложность.

В ряде приложений (таких как высокоскоростные сети или дисковые массивы с поддержкой механизмов безопасности) требуется высокая пропускная способность используемых реализаций хэш-функций, достижимая только для аппаратных решений. Определяющим параметром производительности таких реализаций является глубина схемы. Под глубиной понимается длина максимального простого пути схемы. Вторичным параметром оптимизации является сложность, то есть общее число элементов схемы. Рассматривается базис из элементов конъюнкции, дизъюнкции, отрицания и задержки (при этом отрицание игнорируется при вычислении глубины и сложности).

Предлагаемые методы позволяют существенно понижать глубину схем, реализующих рассматриваемые алгоритмы хэширования. Применение этих методов позволяет получать схемы с глубиной, меньшей, чем у известных реализаций. Часть используемых методов явно или неявно рассматривалась в литературе [3, 4, 5]. Однако в случае явного рассмотрения делались предположения, существенно понижающие общность и как следствие завышающие теоретические нижние оценки глубины [3, 4]. Предлагаемые схемы имеют глубину, меньшую,

чем соответствующие нижние оценки. Схема в работе [5], превосходящая реализацию в работе [3], имеет на один ярус элементов больше, чем предлагаемая реализация.

**Методы оптимизации**

Основными использованными методами оптимизации являются диагональный разрез (в работах [3] и [4] он строился с помощью анализа графа потока данных), симплификация цепочки сумматоров, спекулятивные вычисления и перестановка сумматора и циклического сдвига. Дополнительная оптимизация была достигнута благодаря тождественным преобразованиям булевых функций и разложению булевых функций по переменной.

Диагональный разрез заключается в скашивании границ циклов (этот прием часто применяется при оптимизирующей компиляции программ). Содержательно это можно проиллюстрировать следующим образом. Если изобразить вычисление хэш-функции вертикальной полосой, то вместо стандартного разбиения на прямоугольники, соответствующие раундам преобразования, осуществляется разбиение этой полосы на параллелограммы (по сути путем смещения элементов задержки по схеме) для минимизации длины максимального пути. Возникающие при этом снизу и сверху дополнительные треугольники могут быть включены в схему регулярным образом (иными словами, дополнены до параллелограммов) за счет задания корректных входных значений при инициализации.

Симплификация цепочки сумматоров основана на следующей известной идее. Сумма $x + y + z$ может быть вычислена как сумма $A+B$, где $A = x \oplus y \oplus z$, $B = ShL^1(Maj(x, y, z))$. Это преобразование уменьшает как глубину, так и сложность схемы.

Спекулятивные вычисления также часто применяются при оптимизирующей компиляции программ. Выигрыш здесь может заключаться в том, что вычисление результата прохождения ветви и вычисление значения, определяющего выбор ветви, могут осуществляться параллельно. При схемной реализации алгоритма SHA-1 применение этого приема при вычислении функций $f_t$ позволило получить схему с глубиной, равной глубине сумматора.

Идея перестановки сумматора и циклического сдвига заключается в следующем. Вычисление $ShL^k(A+B)$ можно заменить на эквивалентное выражение $ShL^k(A)+ShL^k(B)+C$, где $C$ — один из элементов четырехэлементного множества $\mathcal{C}_k$, зависящего только от $k$. Выбор элемента определяется значением битов переноса $A+B$. За счет описанной перестановки при реализации MD5 удалось удлинить цепочку сумматоров и понизить глубину благодаря симплификации удлиненной цепочки.

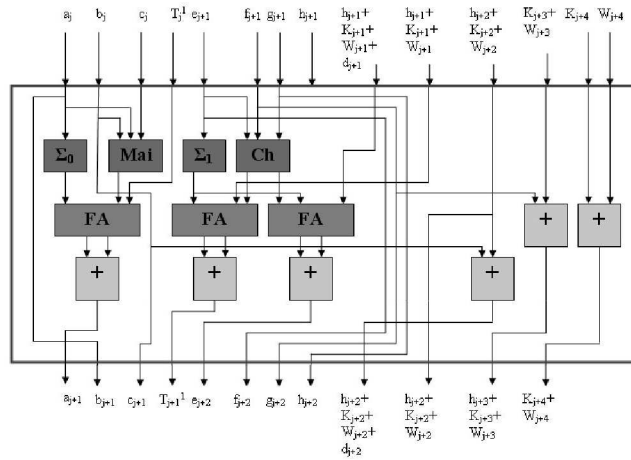


Рис. 1. Реализация алгоритмов семейства SHA-2.

**Схемные реализации**

Применение описанных методов при реализации функций семейства SHA-2 позволило получить схему (рис. 1) глубиной $6+D(n)$, где $D(n)$ — глубина сумматора соответствующей ширины (32 для SHA-224 и SHA-256, 64 для SHA-384 и SHA-512). При этом блок Maj реализуется с глубиной 2 за счет разложения по первой переменной и предвычисления конъюнкции и дизъюнкции второго и третьего аргумента. Стандартная реализация блока Maj увеличивает приведенную оценку глубины на 1.

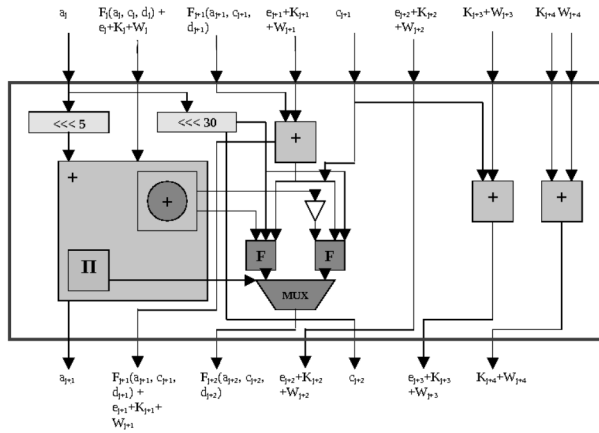


Рис. 2. Реализация алгоритма SHA-1.

Применение описанных методов при реализации функции SHA-1 позволило получить схему (рис. 2) глубиной $D(32)$.

Применение описанных методов при реализации функции MD5 позволило получить схему (рис. 3) глубиной $D(32) + 10$.

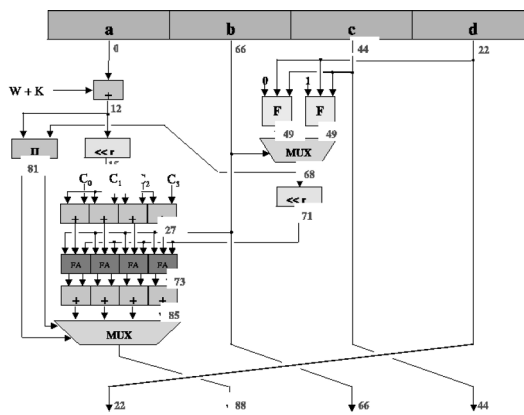


Рис. 3. Реализация алгоритма MD5.

**Список литературы**

[1] NIST, FIPS PUB 180-2. — 2001.

[2] Rivest R. The MD5 Message-Digest Algorithm. — RFC 1321. 1992.

[3] Lee Y. K. et al. Hardware design for Hash functions // Secure Integrated Circuits and Systems, Integrated Circuits and Systems. — 2010. — P. 79–104.

[4] Lee Y. K. et al. Design Methodology for Throughput Optimum Architectures of Hash Algorithms of the MD4-class // Journal of Signal Processing Systems. — 2008. 53 (1–2). — P. 89–102.

[5] Dadda L. et al. The Design of a High Speed ASIC Unit for the Hash Function SHA-256 (385, 512) // DATE'04. — 2004. — P. 70–75.

# О восстановлении параметров $\varepsilon$-безопасности

**Галатенко А. В.**

*agalat@msu.ru*

Напомним определение $\varepsilon$-безопасного языка, введенное в работе [1]. Под конечным автоматом мы будем понимать четверку $V = (A, Q, \varphi, q_0)$, где $A$ — конечное множество входных символов, $Q$ — конечное множество состояний, $\varphi : A \times Q \to Q$ — функция переходов, $q_0 \in Q$ — начальное состояние. Пусть $Q = S \cup I$, причем $S \cap I = \varnothing$. Состояния из $S$ назовем безопасными, состояния из $I$ — небезопасными. Далее будем предполагать, что начальное состояние является безопасным, все состояния достижимы из начального, а $|Q| > 1$.

Обозначим через $A^*$ множество всех конечных слов в алфавите $A$. Функция $\varphi$ может быть продолжена на множество $A^* \times Q$ по мультипликативности.

Подмножество $A^*$ называется языком. Каждому слову $\alpha \in A^*$ соответствует слово $\kappa(\alpha) \in Q^*$, $\kappa(\alpha) = \varphi(\alpha, q_0)$. Рассмотрим произвольное $\varepsilon > 0$. Введем функции $s : Q^* \to \mathbb{N} \cup \{0\}$ и $i : Q^* \to \mathbb{N} \cup \{0\}$ следующим образом. Функция $s(\kappa)$ равняется числу букв $\kappa$, содержащихся в $S$, $i(\kappa)$ равняется числу букв $\kappa$, содержащихся в $I$. Обозначим через $|\kappa|$ число букв в слове $\kappa$. Назовем слово $\kappa$ $\varepsilon$-безопасным, если $\frac{i(\kappa)}{|\kappa|} \leqslant \varepsilon$. Назовем язык $\mathcal{A}$ $\varepsilon$-безопасным ($S_\varepsilon$-языком), если все слова из $\mathcal{A}$ $\varepsilon$-безопасны, и не существует $\varepsilon$-безопасных слов, не принадлежащих $\mathcal{A}$. В работе [1] показано, что $\varepsilon$-безопасные языки являются контекстно-свободными и вообще говоря не регулярными; если $\varepsilon_1, \varepsilon_2 \in \mathbb{Q}$, $0 \leqslant \varepsilon_1 < \varepsilon_2 \leqslant 1$, то существует язык, являющийся $S_{\varepsilon_1}$-языком, но не являющийся $S_{\varepsilon_2}$-языком; если $\varepsilon_2$ не равно 1, то существует язык, являющийся $S_{\varepsilon_2}$-языком, но не являющийся $S_{\varepsilon_1}$-языком.

В работе [2] решалась задача восстановления $\varepsilon$-безопасного языка с помощью кратного условного эксперимента при наличии оракула, для каждого входного слова указывающего, является ли слово $\varepsilon$-безопасным. Были найдены необходимые и достаточные условия восстановления языка с помощью конечного эксперимента при известном разбиении множества состояний и неизвестном значении $\varepsilon$. В частности, если значение $\varepsilon$ таково, что в диаграмме Мура автома-

та имеется пара циклов, причем в одном цикле доля небезопасных состояний больше $\varepsilon$, в другом — меньше, и циклы соединены ориентированным путем, восстановление $\varepsilon$-безопасного языка с помощью конечного эксперимента невозможно.

Рассмотрим обратную задачу. Пусть значение $\varepsilon$ известно, и требуется восстановить разбиение множества состояний. Имеет место следующий результат.

**Теорема.** *Для любого автомата $V$ и любого рационального значения $\varepsilon$, $0 \leqslant \varepsilon \leqslant 1$, существует конечный кратный условный эксперимент, восстанавливающий задаваемый автоматом $\varepsilon$-безопасный язык.*

Автор выражает глубокую благодарность своему научному руководителю, д.ф.-м.н., проф. В. Б. Кудрявцеву за постановку задачи и внимание к работе.

### Список литературы

[1] Галатенко А. В. Автоматные модели защищенных компьютерных систем // Интеллектуальные системы. — 2007. Т. 11. Вып. 1–4. — С. 403–418.

[2] Галатенко А. В. О восстановлении разбиения безопасности // Интеллектуальные системы. — 2010. Т. 14. Вып. 1–4. — С. 123–136.

# К вопросу построения формальных моделей сбоеустойчивых приложений реального времени

**Галатенко В. А., Костюхин К. А., Шмырев Н. В.**
(Москва, Научно-исследовательский институт системных исследований РАН)
*galat@niisi.msk.ru, kost@niisi.msk.ru, shmyrev@niisi.msk.ru*

Моделирование приложений, функционирующих в реальном масштабе времени, требует наличия формализма, позволяющего описать объекты реального времени и их взаимодействие, унифицировать подходы к описанию вычислений. В этой работе мы построим модель вычислений, описывающую основные аспекты контролируемого выполнения приложений реального времени, включающую в себя следующее:

- Статическую и динамическую верификацию корректности
- Оптимизацию и проверку результатов оптимизации
- Устойчивость к сбоям
- Представление вычисления с различной степенью детализации
- Возможность описания ресурсов

Формальное описание устойчивой системы реального времени приводится в работе [1]. В качестве модели приложения в этой работе используются системы переходов [2], в качестве общей спецификации — логика действий со временем (TLA — Temporal Logic of Actions) [3].

Сбои моделируются с помощью множества «сбойных» действий, которые изменяют состояние так же, как и обычные вычисления. Устойчивость к сбоям обеспечивается проведением корректирующих действий.

Приведем несколько определений:

**Определение 1.** Состояние — отображение набора переменных $Var$ на набор значений $Val$.

$$s : Var \to Val.$$

**Определение 2.** Действие — это некоторый предикат над переменными, их значениями, а также их измененными значениями, которые обозначаются штрихом:

$$x' + 1 \leqslant y.$$

Над бесконечными последовательностями действий $\sigma = \sigma_1, \sigma_2, \ldots$ можно рассматривать временные формулы, которые составлены из булевых операторов и операторов линейной временной логики. Например, $[\Box\phi](\sigma)$ — предикат $\phi$ выполняется для любого суффикса $\sigma$, $[\diamond\phi](\sigma)$ — предикат $\phi$ выполняется хотя бы однажды.

**Определение 3.** Программа — набор, состоящий из следующих компонент:

1) Конечное непустое множество $\overline{v}$ состояний;
2) Подмножество внутренних состояний $\overline{x}$ множества $\overline{v}$;
3) Предикат первоначального состояния $\Theta$, включающий в себя только переменные из $\overline{v}$;
4) Конечный набор $A$ действий над переменными из $\overline{v}$.

**Определение 4.** Вычисление программы $P = (\overline{v}, \overline{x}, \Theta, A)$ — это последовательность состояний $\sigma$, такая что выполняются два условия:

1) $\sigma_0$ удовлетворяет $\Theta$
2) $\sigma_{i+1} = \sigma_i$ или найдется такое $\tau \in A$, что $\sigma_{i+1}, \sigma_i$ удовлетворяет $\tau$.

**Определение 5.** Для программы $P = (\overline{v}, \overline{x}, \Theta, A)$ рассмотрим

$$N_P = \bigvee_{\tau \in A} \tau.$$

Тогда точная спецификация определяется как

$$\Pi(P) = \Theta \vee \Box[N_P]_{\overline{v}}$$

и описывает набор всех разрешенных состояний программы.

**Определение 6.** Внешняя спецификация программы задается как:

$$\Phi(P) = \exists \overline{x}.\Pi(P)$$

и описывает все возможные последовательности внешних состояний системы.

И, наконец, введем определение уточнения программ, позволяющее ввести понятие верификации.

**Определение 7.** Отношение уточнения $P_l \sqsubseteq P_h$ означает что программа $P_l$ корректно реализует $P_h$, то есть отношение

$$\Phi(P_l) \Rightarrow \Phi(P_h)$$

выполнено для внешних спецификаций.

### Моделирование разных аспектов вычисления

Естественно описывать вычисление несколькими моделями, отражающими различные аспекты приложения, например, модель памяти и модель занимаемой пропускной способности сети могут быть различными, дополняя основную модель вычислений. Кроме того, набор ограничений и утверждений о приложении может быть достаточно разнородным и задаваться:

1) результатами тестирования;
2) выведенными в процессе анализа приложения утверждениями;
3) проверками во время выполнения;

**Определение 8.** Введем внутреннюю и внешнюю спецификацию набора программ $P_i$:

$$\Pi(P_1, \ldots, P_n) = (\wedge \Theta_i) \vee (\wedge \Box [N_{P_i}]_{\overline{v}}),$$
$$\Phi(P_1, \ldots, P_n) = \exists \overline{x}. \Pi(P_1, \ldots, P_n).$$

**Определение 9.** Будем говорить, что программа $P$ является уточнением набора $P_1, \ldots, P_N$, если

$$\Phi(P) \Rightarrow \Phi(P_1, \ldots, P_n)$$

Таким образом можно определить отношение частичного порядка на множестве наборов программ и использовать его для построения целого семейства моделей, представляющих выполнение приложения с разной степенью детализации.

**Моделирование ресурсов**

Так же, как и время, потребляемые ресурсы важны для доказательства корректности работы приложения и для выполнения его миссии. Для того, чтобы описать потребление ресурсов, также, как и в случае со временем необходимо ввести внутренние переменные, которые описывают ресурс и дополнить соотношения действий границами использования ресурсов.

Например, время моделируется с помощью временных меток действий. Каждое действие $\tau$ связывается с нижней временной границей $L(\tau)$ и верхней временной границей $U(\tau)$. Для того, чтобы считаться успешным, действие должно выполняться по крайней мере $L(\tau)$ временных интервалов. Действие не должно выполняться реже, чем $U(\tau)$.

Определим набор ресурсов $z_i \in Z$ и для каждого из них и для каждого действия определим верхнюю и нижнюю границу потребления ресурсов для каждого действия $U_i(\tau)$ и $L_i(\tau)$.

Для каждого из ресурсов нужно ввести и специальные ограничения, которые описывают их расходование, например, для времени:

$$now = 0 \in \Theta \tag{1}$$

$$\Box[now' \in (now, \infty))]_{now} \tag{2}$$

$$\forall t \in R^+, \diamond(now > t) \tag{3}$$

Для механизма простейшего выделения памяти формулируются более простые условия $\Box mem(\sigma) > 0, mem' > mem - U_m(\tau), mem' < mem - L_m(\tau)$, хотя, для сложных механизмов выделения можно учитывать и более точные характеристики, такие как фрагментацию памяти.

В случае работы с ресурсом нам необходимо расширить определение внутренней и внешней спецификации соответственно:

$$R_P = \bigvee_{\tau \in A} R_i(\tau),$$

$$\Pi(P) = \Theta \vee \Box[N_P]_{\overline{v}} \vee \Box[R_P]_{\overline{v}},$$

где $R_i$ — ограничения, накладываемые на использование ресурса.

**Оптимизация**

Возможность учета понятия оптимизации является естественным требованием к формальной модели приложения. Необходимо рассмотреть две проблемы, возникающие при этом — проблему проверки корректности оптимизирующего преобразования и проблему количественного определения результатов оптимизации. Первая проблема может быть решена с помощью введения эталонной модели вычисления $P_e$, таким образом корректность оптимизации можно определить как соответствие эталонной модели вычисления $P_1 \sqsubseteq P_e \wedge P_2 \sqsubseteq P_e$. Таким образом, для оценки корректности оптимизации нам необходимо зафиксировать эталонную модель вычисления.

Для введения количественных оценок эффективности для последующей оптимизации необходимо ввести внутренние переменные для оптимизации и задать соотношения, описывающие затраты для каждого перехода из состояния $\sigma$ в состояние $\sigma'$ и действия $\tau$ — $C(\tau, \sigma, \sigma')$. Соответственно, изменяется внутренняя спецификация

$$C_P = \bigvee_{\tau \in A, \sigma \in \overline{v}, \sigma' \in \overline{v}} C(\tau, \sigma, \sigma'),$$

$$\Pi(P) = \Theta \vee \Box[N_P]_{\overline{v}} \vee \Box[R_P]_{\overline{v}} \vee \Box[C_P]_{\overline{v}}.$$

**Полная модель приложения**

Таким образом, расширяя описание приложения, мы получаем следующую формальную модель:

**Определение 10.** Программа описывается следующими сущностями:

1) программа без времени $P(\overline{v}, \overline{x}, \Theta, A)$;
2) набор сбоев $F$ и соответствующая программа со сбоями с условием $F(P, F) \sqsubseteq P$;
3) счетчик часов $now$ [1] со свойствами времени 1;
4) Внутренние состояния, описывающие наличие ресурсов, таких как доступная память;
5) функции расхода ресурсов $L_i(\tau)$ и $U_i(\tau)$, задающие границы потребления ресурсов для каждого действия из $P$;

6) Ценовые функции $C(\tau, \sigma, \sigma')$, используемые для оптимизации.

7) Внутренние и внешние спецификации $\Pi(P)$ и $\Phi(P)$.

**Выводы**

В данной работе мы описали формальную модель процесса вычислений, позволяющую описывать и доказывать свойства устойчивых к сбоям вычислений. Эта модель может являться основой для применения формальных методов в среде контролируемого выполнения и обеспечивающей разработку и выполнение встраиваемых приложений [5].

**Список литературы**

[1] Liu Z., Joseph M. Real-Time and Fault-Tolerant Systems. Specification, verification, refinement and scheduling. — UUNU/IIST, 2005.

[2] Pnueli A. The temporal logic of programs // 18th Annual Symposium on Foundations of Computer Science. — 1977. — P. 46–57.

[3] Lamport L. The temporal logic of actions // ACM Transactions on Programming Languages and Systems. — 1994. Vol. 16 (3). — P. 872–923.

[4] Compositional quantitative reasoning / K. Chatterjee, L. de Alfaro, M. Faella et al. // ACM. — 2007.

[5] Вьюкова Н. И., Галатенко В. А., Костюхин К. А., Шмырев Н. В. Организация отладочного комплекса для целевых систем со сложной архитектурой // Информационная безопасность. Микропроцессоры. Отладка сложных систем / под ред. Бетелина. — М.: НИИСИ РАН, 2004. — С. 120–150.

# Исследование групповых свойств умножения с параметром

**Годнева А. В.** (Москва, МГУ им. М. В. Ломоносова)

*god139@yandex.ru*

## Введение

В республике Узбекистан при создании электронной подписи находит широкое применение умножение с параметром. Оно задается парой натуральных чисел $(n, R)$ и определяется следующим образом на элементах $a, b \in \mathbb{Z}_n$:

$$a \circledR b = a + b + abR \,(mod\, n).$$

Заметим, что это коммутативная операция, и любое число, умноженное с параметром на 0, остается неизменным.

Для обоснования безопасности использования умножения с параметром в алгоритме электронной подписи [1] необходимо выяснить возможные порядки элементов относительно возведения в степень с параметром. В случае взаимной простоты $R$ и $n$ это было сделано в работе [2]. Ниже приводится описание групповой части алгебры $(\mathbb{Z}_n, \circledR)$.

## Основной результат

**Теорема.** *Пусть* $n = \prod\limits_{k=1}^{q} p_k^{\alpha_k} * n_0$, $R = \prod\limits_{m=1}^{q'} p_m^{\beta_m} * r_0$, *причем* $R < n$, $(r_0, n) = 1$, $p_k, p_m$ — *простые множители в порядке возрастания, и для любого* $m, 1 \leqslant m \leqslant q'$, $p_m$ *делит* $n$. *Тогда группа обратимых элементов* $\mathbb{Z}_n^{\circledR}$ *алгебры* $(\mathbb{Z}_n, \circledR)$ *представима следующим образом:*

$$\mathbb{Z}_n^{\circledR} \cong \mathbb{Z}_{n_0}^* \times \mathbb{Z}_2 \times \mathbb{Z}_{2^{\alpha_1 - 1}} \times \mathbb{Z}_{p_2^{\alpha 2}} \times \ldots \times \mathbb{Z}_{p_q^{\alpha q}} \text{ при } p_1 = 2 \text{ и } \beta_1 = 1;$$

$$\mathbb{Z}_n^{\circledR} \cong \mathbb{Z}_{n_0}^* \times \mathbb{Z}_{p_1^{\alpha_1}} \times \mathbb{Z}_{p_2^{\alpha 2}} \times \ldots \times \mathbb{Z}_{p_q^{\alpha q}} \text{ в остальных случаях.}$$

**Вспомогательные утверждения**

Обозначим возведение числа $a$ в степень $t$ относительно операции Ⓡ через $a^{\backslash t}$. Пусть $n = \prod\limits_k p_k^{\alpha_k}$, $R = \prod\limits_m p_m^{\beta_m}$, причем простые множители написаны в порядке возрастания. Обозначим $n_i = p_i^{\alpha_i}$, $n = \prod\limits_i n_i$. Если мы решим задачу возведения в степень отдельно для каждого $n_i$, то получим набор $a_i^{\backslash t}$, такой, что $a^{\backslash t} \equiv a_i^{\backslash t}\ (mod\, n_i) \forall i$.

Искомое число найдем, применяя, например, алгоритм Гарнера [3] для решения системы модулярных уравнений.

Если $(n_i; R) = 1$, то формула возведения в степень для обратимого $a$ получена в работе [2]:

$$a^{\backslash t} = \frac{(1 + R * a)^t - 1}{R}\ (mod\, n).$$

В противном случае, будем пользоваться леммой.

**Лемма 1.** *Пусть $n = p^\alpha$, $R = p^\beta * r$, $\beta < \alpha$. Тогда формула для возведения в степень будет выглядеть следующим образом:*

$$a^{\backslash t} = ta + RC_t^2 a^2 + \ldots + R^{k-1} C_t^k a^k,$$

*где $k = \lceil \frac{\alpha}{\beta} \rceil$.*

Доказательство первой леммы несложно провести по индукции с использованием основных свойств биномиальных коэфициентов.

Теперь будем раскладывать группу в произведение циклических. Рассмотрим несколько случаев.

**Случай 1:** $(n, R) = 1$.

**Лемма 2.** *В случае взаимнопростых $n$ и $R$ $\mathbb{Z}_n^{Ⓡ} \simeq \mathbb{Z}_n^*$.*

**Доказательство.** Этот факт доказывается непосредственным построением изоморфизма. Элементу $a \in \mathbb{Z}_n^{Ⓡ}$ сопоставим элемент $\breve{a} \in \mathbb{Z}_n^*$ по формуле $\breve{a} = 1 + Ra$. То, что это отображение является изоморфизмом, проверяется непосредственно, с использованием критерия обратимости в алгебрах с параметром из работы [2] (напомним, что элемент в $\mathbb{Z}_n$ обратим относительно операции Ⓡ тогда и только тогда, когда $(1 + Ra)$ взаимно прост с $n$, то есть обратим в $\mathbb{Z}_n$ по умножению).

Группа $\mathbb{Z}_n i^*$ известным образом раскладывается в произведение циклических (см. [4]).

**Случай 2:** $n = p^\alpha$, $R = p^\beta * r$, $\beta < \alpha$, $p \neq 2$.

В этом случае воспользуемся следующей леммой.

**Лемма 3.** *Пусть* $n = p^\alpha$, $R = p^\beta * r$, $\beta < \alpha$, $p \neq 2$. *Тогда группа* $\mathbb{Z}_n^{\circledR}$ *циклична и 1 является ее образующим элементом.*

Эта лемма доказывается с использованием факта, что порядок элемента делит порядок группы, и проверки, что порядок единицы не может отличаться от порядка группы.

**Случай 3:** $n = p^\alpha$, $R = p^\beta * r$, $\beta < \alpha$, $p = 2$.

Сформулируем еще одну лемму.

**Лемма 4.** *Пусть* $n = p^\alpha$, $R = p^\beta * r$, $\beta < \alpha$, $p = 2$. *Тогда* $\mathbb{Z}_n^{\circledR} \simeq \mathbb{Z}_2 \times \mathbb{Z}_{\frac{n}{2}}$ *или* $\mathbb{Z}_n^{\circledR} \simeq \mathbb{Z}_n$.

Данное утверждение доказывается аналогично предыдущему.

**Доказательство теоремы**

Рассмотрим элемент $a$, обратимый по умножению с параметром, и сопоставим ему кортеж $(a_0, a_1 \dots a_q)$, где

$a_0 \equiv a \ (mod\, n_0)$

$a_1 \equiv a \ (mod\, n_1 = p_1^{\alpha_1})$

...

$a_q \equiv a \ (mod\, n_q = p_q^{\alpha_1})$. Покажем, что введенное отображение является изоморфизмом $\mathbb{Z}_n^{\circledR}$ и прямого произведения $\mathbb{Z}_{n_i}^{\circledR}$ по $i$ от 0 до $q$. Заметим, что все $a_i \in \mathbb{Z}_{n_i}^{\circledR}$. Действительно, это нужно доказывать только для $a_0$, так как в остальных $\mathbb{Z}_{n_i}^{\circledR}$ обратимы все элементы. Предположим, $a_0$ не является обратимым. Тогда $(1 + Ra_0, n_0) \neq 1$. Так как по построению $a = a_0 + n_0 x_0$ для некоторого целого неотрицательного $x_0$, $1 + Ra = (1 + Ra_0) + Rn_0 x_0$, и следовательно $(1 + Ra, n) \neq 1$, что противоречит предположению об обратимости $a$.

В силу взаимной простоты $n_i$ отображение является взаимно однозначным (см. [3]). Остается заметить, что оно выдерживает умножение с параметром. Возьмем обратимое b и разложим его аналогичным образом. Рассмотрим произведение с параметром a и b.

$a \circledR b = a + b + Rab = a_i + x_i n_i + b_i + y_i n_i + R(a_i + x_i n_i)(b_i + y_i n_i) \equiv a_i \circledR b_i \ (mod\, n_i)$.

Значит это искомый изоморфизм из $\mathbb{Z}_n^{\circledR}$ в произведение $\mathbb{Z}_{n_i}^{\circledR}$, а в каждом $\mathbb{Z}_{n_i}^{\circledR}$ разложение в прямое произведение было уже произведено в лемма 2, 3 и 4.

Автор выражает глубокую благодарность своему научному руководителю, н.с. А. В. Галатенко, за постановку задачи и внимание к работе.

**Список литературы**

[1] O'z DSt 1092:2009. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. — Ташкент: Узбекское агентство стандартизации, метрологии и сертификации, 2009.

[2] Ишматова Ю. А. О некоторых свойствах групп алгебр с параметрами // Интеллектуальные системы. — 2012. Т. 16. Вып. 1–4. — С. 291–298.

[3] Василенко О. Н. Теоретико-числовые алгоритмы в криптографии. — М.: МЦНМО, 2003.

[4] Коблиц Н. Курс теории чисел и криптографии. — М.: Научное изд-во ТВП, 2001.

# Математическое ожидание средней длины кодов Хафмана

**Кучеренко Н. С.** (Москва, МГУ им. М. В. Ломоносова)

*nsk.email@gmail.com*

Коды Хафмана и алгоритмы их построения [1] широко применяются при сжатии информации. При построении кода Хафмана полагается, что кодируется алфавит с известными вероятностями появления его символов. На практике значения этих вероятностей могут быть не известны, и для их оценивания необходимо провести предварительную работу, например — вычислить частоту появления символа в сжимаемом файле. Частоту появления символа назовем его *весом.* В силу того, что предварительное вычисление весов может быть трудоемко или невозможно (сжимаемый файл весь не доступен), теоретический интерес представляет следующая задача. Предположим, что веса символов являются случайными величинами, каково тогда математическое ожидание средней длины кода Хафмана.

Обозначим вес символов кодируемого алфавита с мощностью $n$ через $(w_1, \ldots, w_n)$ и положим, что это случайные величины, которые задаются следующим образом. Рассмотрим $x_{(1)} \leqslant x_{(2)} \leqslant \cdots \leqslant x_{(n-1)}$ — вариационный ряд построенный по реализации выборки из закона с функцией распределения $F$, определенной на интервале $(a, b)$. Величину $w_i$ зададим как

$$w_i = x_{(i)} - x_{(i-1)},\ i = 1, \ldots, n,\ x_{(0)} = a, x_{(n)} = b.$$

Обозначим через $T_n^m(F)$ математическое ожидание средней длины кода Хафмана для кодируемого алфавита мощности $n$, кодирующего алфавита мощности $m$ и весов, задаваемых с помощью функции распределения $F$. В работе исследуется поведение функции $T_n^m(F)$ при увеличении мощности кодируемого алфавита $n$ и фиксированных параметрах $m$ и $F$.

Средняя длина кода Хафмана оценивается с помощью энтропии с точностью до константы. Для весов $(w_1, \ldots, w_n)$ и мощности кодирующего алфавита $m$ энтропия $H_m(w_1, \ldots, w_n)$ определяется формулой $-\sum_{i=1}^{n} w_i \cdot \log_m w_i$. При изучении поведения математического

ожидания средней длины кода Хафмана можно перейти к изучению поведения математического ожидания энтропии $H_m(w_1, \ldots, w_n)$ как функции от случайных весов. В работах автора [2–5] применяется похожая техника, поэтому их результаты можно перенести для случая кода Хафмана.

Автором получены достаточные условия на функцию распределения $F$, при которых математическое ожидание средней длины кода Хафмана имеет порядок логарифма (теорема 1), и уточнение этих условий до получения асимптотики (теорема 2).

**Теорема 1.** *Пусть существует невырожденный интервал* $(c, d)$, $(c, d) \subseteq (a, b)$, *и положительные вещественные константы* $c_1$ *и* $c_2$, *для которых выполнено*

$$\forall x, y \in (c, d),\ x > y, \quad c_1 \cdot (x - y) \geqslant F(x) - F(y) \geqslant c_2 \cdot (x - y).$$

*Тогда*

$$T_n^m(F) \asymp \log_m n \quad (n \to \infty).$$

Для формулировки теоремы 2 понадобятся следующие определения. Обозначим носитель функции $f$, определенной на интервале $(a, b)$, через $supp(f)$, $supp(f) = \{x \in (a, b) : f(x) \neq 0\}$. Назовем функцию $f$ *функцией с конечно-интервальным носителем,* если $supp(f)$ имеет вид $supp(f) = \sqcup_{i=1}^{s}(a_i, b_i) \sqcup K$, где $a_i < b_i$, $b_i < a_{i+1}$, $\forall i = 1 \ldots (s - 1)$, $a_s < b_s$, и множество точек $K$ имеет меру ноль.

**Теорема 2.** *Пусть функция распределения* $F$ *представима в виде суммы двух функций — функции скачков* $F'$ *и абсолютно непрерывной функции* $F''$. *Пусть производная* $f$ *функции* $F''$ *имеет конечно-интервальный носитель, ограничена и отделена от нуля на множестве* $supp(f)$ *и интегрируем по Риману, а функция* $F'$ *имеет конечное число скачков. Тогда*

$$T_n^m(F) \sim \log_m n \cdot \int_B 1\, dx \quad (n \to \infty),$$

*где* $B = supp(f)$.

В следующей теореме изучено поведение математического ожидание средней длины кода Хафмана как ограниченной функции от мощности кодируемого алфавита $n$.

**Теорема 3.** *Существуют такая функция распределения $F$, что*

$$\forall n \in \mathbb{N} \qquad T_n^m(F) = 1.$$

*Для любого вещественного $b > 1$ существуют такая функция распределения $F$, что*

$$b + 2 \gtrsim T_n^m(F) \gtrsim b \quad (n \to \infty).$$

Также автором исследован случай когда математическое ожидание средней длины кода Хафмана $T_n^m(F)$ является возрастающей функцией по порядку меньшей, чем логарифм. В теореме 4 описано семейство $S$ возможных асимптотик функции $T_n^m(F)$, а в теореме 5 семейство $S^*$ возможных порядков.

Для описания семейства $S$ понадобится следующее определение. Положительная, возрастающая функция $r(x)$ называется *сохраняющей асимптотику,* если выполнено условие

$$\forall c \in \mathbb{R} \quad r(x + c) \sim r(x) \quad (x \to \infty).$$

Семейство $S$ возможных асимптотик промежуточных функций роста состоит из функций вида $r(\log_m \log_m(n))$, где возрастающая, положительная и дифференцируемая функция $r(x)$, определенная на интервале $(x_0, +\infty), x_0 \geqslant 0$, сохраняет асимптотику и имеет в качестве производной монотонную, положительную и непрерывную функцию $r'(x)$, удовлетворяющую условию:

$$\exists \alpha > 0, \alpha \in \mathbb{R} : \overline{\lim_{x \to +\infty}} \frac{r'(x)}{x^\alpha} < 1.$$

Все функции из $S$ являются возрастающими и имеют порядок меньше чем $\log_m n$ при $n \to \infty$.

**Теорема 4.** *Для любой функции $r(\log_m \log_m(n))$ из семейства $S$ существуют функция распределения $F$ такая, что*

$$T_n^m(F) \sim r(\log_m \log_m(n)) \quad (n \to \infty).$$

Для формулировки следующей теоремы понадобится понятие сохраняющей порядок функции. Положительная, возрастающая функция $r(x)$ называется *сохраняющей порядок,* если выполняется условие

$$\forall c \in \mathbb{R}, c \neq 0, \quad r(c \cdot x) \asymp r(x) \quad (x \to \infty).$$

Семейство $S^*$ возможных порядков промежуточных функций роста состоит из функций вида $r(\log_m(n))$, где неограниченно возрастающая, положительная и дифференцируемая функция $r(x)$, определенная на интервале $(x_0, +\infty)$, $x_0 \geqslant 0$, сохраняет порядок и имеет в качестве производной монотонную, положительную и непрерывную функцию $r'(x)$, удовлетворяющую условию:

$$\exists \alpha \in \mathbb{R}, 0 < \alpha < 1 : \overline{\lim_{x \to +\infty}} \frac{r'(x)}{x^{\alpha-1}} \leqslant 1.$$

Все функции из $S^*$ являются возрастающими и имеют порядок меньше чем $\log_m n$ при $n \to \infty$. В отличие от класса $S$, в классе $S^*$ есть функции по порядку большие чем любая функция из $S$, например $(\log_m n)^\alpha$, $0 < \alpha < 1$. Автором показано, что для функций семейства $S^*$ верна следующая теорема

**Теорема 5.** *Для любой функции $r(\log_m(n))$ из семейства $S^*$ существуют функция распределения $F$ такая, что*

$$T_n^m(F) \asymp r(\log_m(n)) \quad (n \to \infty).$$

**Список литературы**

[1] Яблонский С. В. Введение в дискретную математику. — М.: Высшая школа, 2002.

[2] Кучеренко Н. С. Сложность поиска идентичных объектов в случайных базах данных // Интеллектуальные системы. — 2007. Т. 11. Вып. 1–4. — С. 525–550.

[3] Кучеренко Н. С. О промежуточных функциях роста сложности поиска для случайных баз данных // Интеллектуальные системы. — 2009. Т. 13. Вып. 1–4. — С. 361–395.

[4] Кучеренко Н. С. Задача поиска по ключу с определением позиции // Интеллектуальные системы. — 2010. Т. 14. Вып. 1–4. — С. 293–306.

[5] Кучеренко Н. С. Средняя сложность поиска идентичных объектов для случайных неравномерных баз данных // Дискретная математика. — 2011. Т. 23. Вып. 2. — С. 129–158.

# Об одном подходе к линейной адаптивной цифровой обработке сигналов

**Мазуренко И. Л.** (Москва, МГУ им. М. В. Ломоносова)

*ivan@mazurenko.ru*

Адаптивная линейная система с обратной связью представляет собой адаптивный алгоритм, получающий на вход разность результата обработки входного сигнала $x$ некоторым устройством обработки информации и требуемого выходного сигнала $d$ ($x - d = \varepsilon$ — сигнал ошибки), и итерационно подстраивающий параметры устройства обработки информации с целью минимизации сигнала ошибки [1].

Примерами адаптивных систем являются системы линейного предсказания сигнала, широко применяющаеся при цифровом кодировании речи, системы моделирования и идентификации, применяющиеся для изучения вибраций механических системы, системы выравнивания характеристик, использующиеся для исключения влияния каналов связи, системы подавления шумов и помех, применяющиеся в задачах адаптивной шумоочистки и адаптивного эхоподавления. Рассмотрению алгоритмических подходов к решению последней задачи и посвящена настоящая работа.

В задаче адаптивной линейной фильтрации предполагается, что устройство обработки входной информации — линейно, и адаптивный алгоритм используется для итеративного подстраивания параметров этой линейной системы, а именно коэффициентов $H = (h_0, \dots h_{L-1})$ линейного фильтра с конечной импульсной характеристикой. При этом выходной сигнал $y$ в момент времени $k$ получается путем свертки входного сигнала $x$ и коэффициентов фильтра: $y_k = \sum_{l=0}^{L-1} h_l x_{k-l}$. Наиболее распространенным применением адаптивной линейной фильтрации на практике является задача подавления эхосигнала в телефонных (проводных и беспроводных) сетях. Эхо в таких сетях делится на электрическое (возникающее в так называемых «гибридах» — устройствах преобразования 2-проводных телефонных сетей в 4-проводные) и акустическое (возникающее в оконечном оборудовании: мобильных и стационарных телефонах, устройствах обратной связи, оконечных устройствах IP-телефонии).

Адаптивные алгоритмы подстройки параметров линейного фильтра основываются на принципе минимизации квадрата сигнала ошибки $\varepsilon$. В качестве параметра сигнала ошибки рассматривают оценку величины среднеквадратического отклонения сигнала $E(\varepsilon_k^2) = E(d_k^2) + H^T E(X_k X_k^T) H - 2E(d_k X_k^T) H$.

Поскольку величина среднеквадратического отклонения сигнала ошибки представляет собой положительно определенную квадратичную форму от коэффициентов фильтра $H$, минимум квадрата ошибки достигается в точке равенства нулю градиента $\nabla(\varepsilon_k^2) = 2RH - 2P$, где $R = E(X_k X_k^T)$ — LxL-матрица автокорреляции, $P = E(d_k X_k^T)$ — корреляция эхо и требуемого выходного сигнала $d$. Это дает нам известную теорему Винера-Хопфа [1], позволяющую найти «идеальный» фильтр H, минимизирующий квадрат ошибки предсказания: $H = R^{-1}P$.

Поскольку на практике ни оценка матрицы автокорреляции $R$, ни вектор $P$ точно неизвестны, используют итерационные методы, в той или иной степени сводящиеся к применению метода градиентного спуска: $H_{k+1} = H_k + \mu(-\nabla_k)$, $0 < \mu < \frac{1}{\lambda_{\max}}$, где $\lambda_{\max}$ — максимальное по модулю собственное значение матрицы автокорреляции $R$.

Наиболее простой и распространенной модификацией метода градиентного спуска является так называемый «метод наименьших квадратов», который применительно к данной задаче сводится к тому, что в качестве оценки математического ожидания квадрата ошибки $E(\varepsilon_k^2)$ берется величина $\varepsilon_k^2$. Формула адаптивного обновления оценки коэффициентов фильтра получается значительно более простой вычислительно: $H_{k+1} = H_k - \mu\hat{\nabla}_k = H_k + 2\mu\varepsilon_k X_k$, $0 < \mu < tr(R)$, ибо требует $2L + 2$ умножений. Более быстро сходящийся алгоритм нормализованных наименьших квадратов использует $2L + 2$ умножений и одно деление: $H_{k+1} = H_k + 2\mu\varepsilon_k \frac{X_k}{\|X_k\|^2}$, $0 < \mu < 1$.

В конце XX – начале XXI века было предложено несколько алгоритмов цифровой обработки сигналов, более эффективных с точки зрения соотношения скорости сходимости и вычислительной сложности. К их числу можно отнести метод аффинных проекций [2], метод быстрых аффинных проекций ([3,4]) и приближенные методы быстрых аффинных проекций, основанные на Теплицевом приближении оценки автокорелляционной матрицы $R$ ([5] и др.).

Метод аффинных проекций, обладающий наивысшей из известных методов скоростью сходимости, имеет сложность $2LN + K_{inv}N^2$ умножений, где $K_{inv}$ — сложность обращения матрицы, а потому практически неприменим на практике. Его упрощение — метод быстрых аффинных проекций — обладает уже линейной сложностью относительно размера проекции $N$ и числа коэффициентов фильтра $L$ — $2L + 20N$ умножений, — однако, не лишен недостатков, один из которых (неустранимое накопление ошибок при целочесленной реализации) делает его практически неприменимым. Приближенные методы алгоритма быстрых аффинных проекций, описанные [5], лишены данного недостатка и дают сложность в лучшем случае $2L+8N-3$ умножений, однако не обладают стабильностью в случае быстроменяющегося по своим спектральным характеристикам входного сигнала $x$.

Автором данной работы была предложена модификация приближенного метода быстрых аффинных проекций, основанного на применении Теплицевого приближения автокорреляционной матрицы $R$, в котором, в отличие от алгоритма [5], на каждом шаге итерации алгоритма Левинсона-Дурбина испольнуется новое приближение автокорреляционной матрицы, а для контроля за сходимостью алгоритма применяется пороговое правило контроля уровня недекоррелированной ошибки адаптации. Данная модификация алгоритма, практически не проигрывая в скорости методу [5], дает на тестовых данных значительно более высокую скорость сходимости и надежность работы алгоритма адаптации. Работа защищена патентом США.

**Список литературы**

[1] Уидроу Б., Стирнз С. Адаптивная обработка сигналов. — М.: Радио и связь, 1989.

[2] Ozeki K., Umeda T. An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties // Proc. of the Elec. Comm. Japan. Vol. J67-A. — February 1984. — P. 126–132.

[3] Gay S. L. A fast converging, low complexity adaptive filtering algorithm // Third International Workshop on Acoustic Echo Control. — 7–8 Sept. 1993. Plestin les Greves, France.

[4] Tanaka M., KanedaY., Makino. S., Kojima J. A fast projection algorithm for adaptive filtering // IEICE Trans. Fund. E78-A (10m). — October 1995.

[5] Ding H. Fast affine projection adaptation algorithms featuring stable symmetric positive-definite linear system solvers // Applications of Signal Processing to Audio and Acoustics. — 2005.

## Решение задачи оптимизации безопасной информационной системы

**Марков А. С., Патраков Н. В., Фадин А. А.** (Москва)

*nikolay.patrakov@gmail.com, fadin.andrey@gmail.com*

Построение безопасных информационных систем является необходимым условием для функционирования всех современных государственных, общественных и коммерческих организаций.

Что следует понимать под безопасной информационной системой? В соответствии с Национальным стандартом Российской Федерации «Информационная технология. Практические правила управления информационной безопасностью» (ГОСТ Р ИСО/МЭК 17799–2005), информационная безопасность — защита конфиденциальности, целостности и доступности информации. Здесь, конфиденциальность — это свойство информационных ресурсов, в том числе информации, связанное с тем, что они не станут доступными и не будут раскрыты для неуполномоченных лиц. Целостность — это неизменность информации в процессе ее передачи или хранения. Доступность — это свойство информационных ресурсов, в том числе информации, определяющее возможность их получения и использования по требованию уполномоченных лиц.

Фактически обеспечением безопасности для информационной системы является выполнение приведенных выше трех свойств.

Что является оптимизацией системы? Изменение состава или структуры частей (компонентов) системы для достижения оптимального значения для одного или нескольких свойств системы с сохранением допустимых значений для всех остальных свойств.

В данной работе поставлена задача по построению компонентной модели информационной системы, которая:

1) подходит для оценки свойств системы, связанных с безопасностью;
2) удобна для выполнения преобразований с целью ее дальнейшей оптимизации.

Система — это множество элементов, находящихся в связях и отношениях друг с другом.

Для информационной системы предлагается использовать следующий подход.

Вводятся две различные сущности: компонент и интерфейс.

Соответственно безопасность системы зависит от выполнения:

1) требований по доверию (assuarance) ко всем ее компонентам и интерфейсам;
2) требований по функционалу ко всем ее компонентам и интерфейсам.

Математической моделью данной информационной системы будет являться ориентированный граф, вершины которого представляют собой компоненты безопасной информационной системы, а рёбра — интерфейсы и каналы коммуникации между ними.

Компоненты могут быть:

1) высокоуровневые сущности: СУБД, операционная система, вебсервер, коммутационное устройство сети, ЭВМ;
2) сущности более низкого уровня: библиотеки, исполняемые модули, брокеры запросов удаленного вызова процедур;
3) сущности связанные с внешними системами и ролями пользователей: администратор, злоумышленник, клиентская машина. Как правило, в рамках моделирования их внутренний состав не рассматривается.

Для снижения числа связей и упрощения дальнейшего анализа принимается решение разделить все компоненты на два вида:

1) функциональные — связанные непосредственно с назначением системы;
2) инфраструктурные — обеспечивающие среду выполнения и установление коммутации между всеми остальными компонентами.

Безусловно, эта граница не всегда является четкой, некоторые компоненты могут сочетать в себе свойства двух типов, но инфраструктурные компоненты (гипервизор, драйвер файловой системы, брокер запросов CORBA) характеризует в целом большое количество однотипных связей, если же рассматривать их функционирование, то, как правило, они обрабатывают данные других компонентов и редко бывают инициаторами запросов.

Фактически, если рассматривать систему, любые два функциональных компонента взаимодействуют через инфраструктурный компонент, соответственно свойства (и требования по безопасности) для всех рёбер между функциональными компонентами можно сопоставить с необходимыми свойствами инфраструктурных элементов.

Очевидно, что в дальнейшем для оценки свойств системы достаточно исследования свойств вершин соответствующего ей графа.

Каждый элемент может быть связан с другим по двум основным требованиям: по чтению/выполнению и по записи. Кроме того, вводится метрика критичности связи, то есть степень важности этой связи для зависимого компонента.

В первом случае возможно нарушение доступности в зависимом компоненте по результатам операций во влияющем компоненте. Во втором случае возможно нарушение целостности, доступности и конфиденциальности.

Дальнейший анализ системы целесообразно выполнять на основе матрицы, построенной по описанному выше графу.

Перечень возможных свойств объектов системы:

1) наименование компонента;
2) автор компонента;
3) срок существования (сколько времени пришло с начало первого внедрения и его использования);
4) количество найденных уязвимостей в компоненте за всё время его использования;
5) перечень типов данных ограниченного пользования, которые обрабатывает компонент;
6) требования по доступности компонента.

**Список литературы**

[1] Марков А. С., Цирлов В. Л. Направления совершенствования методов сертификации программного обеспечения на отсутствие уязвимостей // Ежегодное совещание «Взаимодействие участников рынка продуктов и услуг в области безопасности». — М., 2007.

[2] Ghosh A. K., McGraw G. An Approach for Certifying Security in Software Components // Reliable Software Technologies report. — 2001.

# Атаки на протокол Нидхема-Шредера. Модификация протокола Нидхема-Шредера и построение на его основе

**Мозгалева О. А.** (Москва, МГУ им. М. В. Ломоносова)

*omozgaleva@gmail.ru*

В настоящее время одним из важнейших сервисов информационной безопастности является аутентификация. Весьма известным решением проблемы аутентификации является протокол Нидхема-Шредера (R.M. Needham, M.D. Schroeder)[1], предложенный в Нидхемом и Шредером в 1978 году. Этот протокол использует для аутентификации третью доверенную сторону, формирующую сеансовый ключ для идентификации одной стороны перед другой.

Протокол аутентификации Нидхема-Шредера выглядит следующим образом.

1. Абонент $A$ вырабатывает случайное $r_A$ и отправляет центру $S$ запрос $m_A = (idA, idB, r_A)$.

2. Центр S вырабатывает сеансовый ключ $K$ и отправляет абоненту $A$ сообщение $\omega_S = E_{K_AS}(r_A, idB, K, E_{K_BS}(K, idA))$, зашифрованное ключом $K_AS$, разделенным между центром $S$ и абонентом $A$.

3. Абонент $A$ расшифровывает сообщение $\omega_S$ и проводит аутентификацию центра $S$ по параметру $r_A$. В случае успешной аутентификации абонент $A$ пересылает абоненту $B$ сообщение $y_B = E_{K_BS}(K, idA)$.

4. Абонент $B$ расшифровывает сообщение $y_B$, на основе знания ключа $K_BS$, разделенного им с центром $S$, извлекая при этом сеансовый ключ $K$. Абонент $B$ вырабатывает случайный параметр $r_B$, и отправляет сообщение $\omega_B = E_K(r_B)$ абоненту $A$.

5. Абонент $A$ расшифровывает сообщение $\omega_B$ на ключе $K$, определяет $r_B$, формирует и отправляет абоненту $B$ сообщение $\omega_A = E_K(\psi(r_B))$, где $\psi$ — некоторая заранее выбранная числовая функция.

6. Абонент $B$ расшифровывает сообщение $\omega_A$, вычисляет свое $\psi(r_B)$ и сравнивает результаты, тем самым аутентифицируя абонента $A$.

Протокол Нидхема-Шредера является самым известным протоколом аутентификации, однако он уязвим для атаки, изобретенной в 1981 году Деннингом (Denning) и Сакко (Sacco)[3]. Эта атака основана на атаке с повторной передачей сообщения. Предполагается, что произошла компроментация сеансового ключа. Повторно отправив соответствующее этому ключу сообщение, злоумышленник может аутентифицироваться под видом законного абонента.

Мною был изучен протокол Нидхема-Шредера и получены следующие результаты.

Первый результат связан с ограничениями на функцию шифрования. При определенных свойствах функции шифрования $E$ один абонент сможет аутентифицироваться у другого абонента под видом некоторого третьего законного абонента.

**Теорема 1.** *Если в протоколе Нидхема-Шредера*

*1. Функция шифрования является гомоморфизмом относительно пар (K,id): $E_{K_BS}(K_1 \bullet K_2, idA_1 \circ idA_2) = E_{K_BS}(K_1, idA_1) * E_{K_BS}(K_2, idA_2)$, где «•», «∘», «∗» — групповые операции на соответствующих множествах.*

*2. Абоненту A известно разложение идентификатора законного абонента C относительно групповой операции: $idC = idA^{\alpha} \circ idA_1{}^{\alpha_1} \circ \ldots idA_n{}^{\alpha_n}$, где $A_i$ — законные участники. То абонент A с помощью абонентов $A_1, \ldots, A_n$ сможет аутентифицироваться у абонента B под видом законного абонента C.*

Второй результат показывает необходимость выбора параметра $r_A$ случайным для предотвращения возможности атаки, совмещающей в себе атаку «противник в середине» и атаку с повторной передачей сообщения. Данный результат очень важен, так как открывает большие возможности взлома протокола при успешных атаках на генератор случайных чисел.

**Теорема 2.** *Если параметр $r_A$ фиксирован, и происходит компроментация сеансового ключа K (использованного ранее в других сеансах или текущего), то становится возможной атака со стороны злоумышленника C совмещающей в себе атаку «противник в середине» и атаку с повторной передачей сообщения. В результате такой атаки:*

*1. Злоумышленник $C$ аутентифицируется у абонента $B$ под видом абонента $A$.*

*2. Более того, злоумышленник $C$ сможет «подслушивать» все дальнейшие сообщения, передаваемые абонентом $A$ абоненту $B$, и наоборот.*

Первое утверждение теоремы очевидным образом следует в результате атаки Деннинга-Сакко. Второе утверждение получается в результате следующей атаки. На шаге 2 злоумышленник $C$ перехватывает новое сообщение $\omega'_S$ от центра аутентификации $S$ к абоненту $A$, и подменяет его старым сообщением $\omega_S = E_{K_AS}(r_A, idB, K, E_{K_BS}(K, idA))$, соответствующим старому ключу $K$. Так как абонент $A$, аутентифицирует центр $S$, расшифровывая сообщение $\omega_S$, по параметру $r_A$, который в свою очередь постоянен по условию теоремы, то аутентификация центра $S$ пройдет успешно и участник $A$ продолжит выполнение протокола. Таким образом, все последующие сообщения информационного взаимодействия абонентов $A$ и $B$, будет происходить на основе шифрования ключом $K$, который в свою очередь известен злоумышленнику $C$, а, значит, он сможет «подслушивать» все сообщения абонентов $A$ и $B$, которые даже не будут это подозревать.

Следующий результат представляет собой модификацию протокола Нидхема-Шредера. Полученная модификация, во-первых, позволяет предотвращать атаку с повторной передачей сообщения. Это реализовывается введением временной метки $T$. Во-вторых, она позволяет строить другие блочные протоколы, использующие несколько дополнительных серверов. Для этого вводится несколько заглушек, для последующего связывания нескольких модифицированных протоколов в один блочный протокол.

Модифицированная схема Нидхема-Шредера:

$$N - Sch - mode(r, idA_1, idS, idA_2, K_{A_1S}, K_{A_2S}, T, id1, id2, T', K', K'', r') = (K)$$

Входные данные: $r$ — случайное число, $idA_1, idS, idA_2$ — идентификаторы абонентов и сервера аутентификации, $K_{A_1S}, K_{A_2S} K_{A_1S}, K_{A_2S}$ — долговременные ключи, распределенные между сервером аутентификации и абонентами, $T$ — время действия нового сеансового ключа.

Заглушки: $id1, id2$ — идентификаторы законных пользователей, $T'$ — временная метка, $K', K''$ — ключи, $r'$ — случайное число. Выходные данные: $K$ — сеансовый ключ.

1. Абонент $A_1$ вырабатывает случайное $r$ и отправляет серверу $S$ сообщение $m_s = (r, idA_1, idA_2, id1, id2, T', K')$.

2. Сервер $S$ отправляет абоненту $A_1$ сообщения
$x_S = E_{K_{A_1S}}(r, idA_2, id1, K, T)$, $x_{A_2} = E_{K_{A_2S}}(idA_1, idA_2, id1, K, T)$.

3. Абонент $A_1$ вырабатывает случайное $r'$ и отправляет абоненту $A_2$ сообщение $m_{A_2} = (r', Aut = E_K(idA_1, T, K''), x_{A_2})$.

Время $T$ — это время действия сеансового ключа $K$. Соответственно теперь противник сможет посылать повторно сообщение лишь в период времени $T$. Соответственно $T$ должно быть выбрано таким, чтобы за это время компроментация ключа $K$ была возможна с очень маленькой вероятностью.

Важным следствием этой модифицированной схемы является ее связь с протоколом Kerberos, который является самым актуальным и распространенны протоколом сетевой аутентификации. Протокол Kerberos представляется прямой суммой двух модифицированных схем Нидхема-Шредера с завершающей посылкой, для взаимной аутентификации. Более того в результате последовательного выполнения нескольких модфицированных схем Нидхема-Шредера можно получить протокол, использующий несколько вспомогательного сервера. Этот протокол в сравнении с модифицированным протоколом Нидхема-Шредера (однократным его применением) имеет важные преимущества. Во-первых, в очень больших сетях данный протокол позволяет уменьшить нагрузку на сервера, распределяя ее среди серверов, обслуживающих маленькие подсети. Во-вторых, позволяет ускорить процесс повторной аутентификации.

**Список литературы**

[1] Гашков С. Б., Применко Э. А., Черепнев М. А. Криптографические методы защиты информации / (1-е изд.) Учеб. пособие. — ИЦ Академия, 2010. ISBN: 978–5–7695–4962–5.

[2] Kohl J., Neuman C. The Kerberos Network Authentication Service (V5). RFC 1510. — September 1993.

[3] Denning D. E., Sacco G. M. Timestamps in key distributed protocols // Communication of the ACM. — 1981. 24 (8). — P. 533–535.

# О поляризации источников Бернулли случайными линейными преобразованиями

**Пантелеев П. А.** (Москва, МГУ им. М. В. Ломоносова)

*panteleev@intsys.msu.ru*

В недавней работе Э. Арикана [1] построен класс эффективно кодируемых/декодируемых кодов, достигающих границы Шеннона для двоичного симметричного канала. По существу, как было отмечено в [2], идея построения полярных кодов опирается на *феномен поляризации* дискретных вероятностных источников, состоящий в том, что после применения некоторого специально подобранного преобразования к последовательности $X_1, \ldots, X_n$ двоичных случайных величин получается последовательность двоичных случайных величин $Y_1, \ldots, Y_n$, которую можно условно разбить на две компоненты — «случайную» $Y_{i_1}, \ldots, Y_{i_m}$ и «детерминированную» $Y_{j_1}, \ldots, Y_{j_k}$. Энтропия случайной компоненты близка к максимально возможной, а детерминированная компонента почти однозначно восстанавливается по случайной. Данное обстоятельство позволяет использовать такие преобразования для порождения равномерно распределенных псевдослучайных последовательностей [2]. В настоящей работе изучается феномен поляризации для случайных линейных преобразований.

Для произвольных дискретных случайных величин $U$ и $V$ энтропию $U$ будем обозначать через $H(U)$, а условную энтропию $U$ при условии $V$ через $H(U \mid V)$. Рассмотрим произвольную последовательность двоичных случайных величин $\mathbf{X} = X_1, X_2, \ldots$, которую мы будем называть *источником*. Положим $\mathbf{X}^{(n)} = (X_1, \ldots, X_n)$. Если существует предел

$$\lim_{n\to\infty} \frac{1}{n} H\left(\mathbf{X}^{(n)}\right),$$

то будем называть его *энтропией источника* $\mathbf{X}$, и обозначать $h(\mathbf{X})$. Важным частным видом вероятностного источника является *источник Бернулли*, у которого все случайные величины $X_i$ независимы в совокупности и имеют одинаковое распределение Бернулли с параметром $p$. Легко видеть, что для любого такого источника $\mathbf{X}$ энтропия существует и равна $h(\mathbf{X}) = -p\log_2 p - (1-p)\log_2(1-p)$.

Пусть $\mathbb{F}_2^n$ — множество всех двоичных векторов длины $n$. Зададим произвольную последовательность отображений $g_n\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$, $n \in \mathbb{N}$, преобразующих случайный двоичный вектор $\mathbf{X}^{(n)} = (X_1, \ldots, X_n)$ в случайный двоичный вектор $\mathbf{Y}^{(n)} = (Y_1, \ldots, Y_n) = g_n(\mathbf{X}^{(n)})$. Для каждого $\varepsilon > 0$ рассмотрим величину

$$h_\varepsilon(\mathbf{Y}^{(n)}) = \frac{1}{n}|\{i \in \{1, \ldots, n\} \mid H\left(Y_i \mid \mathbf{Y}^{(i-1)}\right) > 1 - \varepsilon\}|.$$

Пусть $i_1, \ldots, i_m$ есть в точности все индексы $i \in \{1, \ldots, n\}$ для которых выполняется $H\left(Y_i \mid \mathbf{Y}^{(i-1)}\right) > 1 - \varepsilon$. Тогда компонента $Y_{i_1}, \ldots, Y_{i_m}$ случайного вектора $\mathbf{Y}^{(n)}$ имеет энтропию $H\left(Y_{i_1}, \ldots, Y_{i_m}\right)$, отличающуюся от максимально возможной не более чем в $(1-\varepsilon)$ раз, а величина $h_\varepsilon(\mathbf{Y}^{(n)})$ есть доля этой компоненты в $\mathbf{Y}^{(n)}$.

Скажем, что последовательность отображений $g_{i_1}, g_{i_2}, \ldots$ *поляризует* источник $\mathbf{X}$ если для сколь угодно малого $\varepsilon > 0$ выполняется

$$\lim_{k\to\infty} h_\varepsilon(\mathbf{Y}^{(i_k)}) = h(\mathbf{X}).$$

Как показано в [1, 2], если в качестве преобразования $g_{2^k}$ берется умножение на $2^k \times 2^k$-матрицу $\mathbf{G}_k = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)^{\otimes k}$, где $\otimes k$ — есть $k$-ая кронекерова степень матрицы, то последовательность $g_1, g_2, g_4, \ldots$ поляризует произвольный источник Бернулли.

Рассмотрим ансамбль $\mathcal{A}_n$ случайных двоичных $n \times n$-матриц такой, что все матрицы в нем равновероятны.

**Теорема.** *Пусть* $\mathbf{A}_n$ *— случайная матрица из ансамбля* $\mathcal{A}_n$*, а* $\mathbf{X}$ *— произвольный источник Бернулли. Тогда для любого сколь угодно малого* $\delta > 0$ *и* $\varepsilon_n = o(1)$ *выполняется*

$$\lim_{n\to\infty} \Pr\left[|h(\mathbf{X}) - h_{\varepsilon_n}(\mathbf{A}_n\mathbf{X}^{(n)})| < \delta\right] = 1.$$

Таким образом, при растущем $n$ почти все линейные преобразования поляризуют источники Бернулли.

**Список литературы**

[1] Arikan E. Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels. IEEE Trans. on Inf. Theory. 2009. V. 55. Iss. 7. P. 3051–3073.

[2] Abbe E. Randomness and dependencies extraction via polarization. Inf. Theory and Apps Workshop. 2011. La Jolla, CA. P. 1–7.

## Быстрое умножение матриц над полем из двух элементов

**Чечулина К. А.** (Москва, МГУ им. М. В. Ломоносова)

*chechulina.karina@gmail.com*

В данной работе речь пойдет о блочных алгоритмах умножения матриц над полем из двух элементов и о параллельных вариантах этих алгоритмов.

Для умножения над полем из двух элементов матрицы $D$ размера $N \times n_1$ на блок $M$ размера $n_1 \times n$ , представленный построчно машинными словами, используется алгоритм «четырех русских». Кратко опишем алгоритм.

Разобьем матрицу $D$ на вертикальные блоки с горизонтальным размером $k$. На первом этапе алгоритма составляем $\frac{n_1}{k}$ массивов длины $2^k$, в каждом из которых хранятся произведения всевозможных битовых строк длины $k$ на соответствующую часть блока $M$. Благодаря тому, что битовые строки упорядочены в лексикографическом порядке, каждый элемент получается сложением двух предшествующих элементов из данного массива. С учетом того, что строки блока $M$ являются машинными словами, их сложение занимает одну операцию. Таким образом, на создание всех массивов понадобится $\frac{n_1}{k} \times 2^k$ сложений по модулю 2. На втором этапе умножаем построчно матрицу $D$ на блок $M$, складывая машинные слова, соответствующие каждому блоку из $k$ бит, который берется остатком от деления машинного слова на $2^k$, а само машинное слово заменяется целой частью от того же деления. При такой реализации для умножения одной строки на блок $M$ потребуется $\frac{n_1}{k}$ сложений. Получение значения, хранящегося в каждом блоке, требует две операции, и общее число операций в программе не превысит $\frac{n_1}{k} \times 2^k + 2 \times N \times \frac{n_1}{k}$. Написанные и протестированные программы подтвердили теоретические оценки сложности.

Рассмотрим параллельные варианты данного алгоритма. Сначала остановимся на параллельной программе умножения матриц машинных слов, рассчитанной на машины с общей памятью на все ядра. В таком случае матрицу $D$ делим на равные вертикальные блоки, ядру будет соответствовать некоторое количество столбцов. Далее каждое

ядро умножает свою часть матрицы $D$ на соответствующую часть столбца $M$, получая столбец — произведение. Все эти столбцы необходимо сложить. Время работы оценивается величиной $\frac{2\times N\times n_1}{k\times nit} + N$ операций, где $nit$ — количество ядер. Протестированные на СКИФ МГУ программы показали, что на машинах с общей памятью достигается почти 100% ускорение, что соответствует теоретическим оценкам, поскольку не зависящее от $nit$ слагаемое мало.

Второй вариант программы рассчитан на вычислительные узлы с разделенной памятью. Матрица $D$ делится между вычислительными узлами горизонтально, то есть каждый узел получает примерно одинаковое количество строк, которые умножает на столбец $M$, получая соответствующие строки матрицы-произведения. Далее, с уменьшенным параметром $N$, каждый узел делит между своими ядрами абсолютно так же, как в предыдущем варианте программы (вертикально). После этого все результаты надо циклически переслать остальным вычислительным узлам. В таком случае каждый вычислительный узел тратит $\frac{n_1\times 2^k}{nit\times k} + \frac{N\times n_1}{nit\times k\times p}$ операции на нахождение своей части матрицы, где $p$ — количество вычислительных узлов. Если рассматривать только вычислительную часть программы, то достигается 100% ускорение как в зависимости от числа вычислительных узлов, так и в зависимости от числа ядер на узле. Однако если учитывать и пересылки, то использование большого числа вычислительных узлов не является эффективным, поскольку на сбор информации понадобится значительно больше времени, чем на вычисление. Оптимальным параметром $p$ при больших $N$ является $p = 40$.

Пусть требуется вычислить $L^T \times M$, где $L, M \in \mathbb{F}(N \times s \times n)$. Разделим левый блок $L$ по столбцам на блоки меньшего размера $k$. В каждом блоке записаны слова из $k$ бит, поэтому различных строк не более $2^k$. Обозначим блоки через $L_i$ и будем умножать $L_i^T$ на блок $M$, в каждой строке которого $s$ машинных слов. На первом этапе проходим блок $L_i$ слева направо все столбцы и выбираем разные. Если $j$-ый столбец уже встречался на месте $t$, то прибавляем $j$-ую строку блока $M$ к $t$-ой строке. Далее рассматриваем только различные столбцы блока $L_i$. На втором этапе получившиеся матрицы размера не более чем $2^k$ перемножаем. Число операций на этом этапе равно

$n \times s^2 \times 2^k$. Выбирая $k = \log_2 N - \log_2 \log_2 N$, получаем общую оценку сложности работы программы $3 \times \frac{N \times n \times s^2}{\log_2 N - \log_2 \log_2 N}$. Проведенные тесты подтвердили теоретические оценки.

Первые два способа распараллеливания написаны для машин с общей памятью на все ядра.

Первый способ заключается в том, что каждому ядру выделяется равное количество вертикальных блоков $L_i$. Однако всем ядрам придется скопировать себе весь блок $M$. Как и в первом варианте распараллеливания умножения матриц, блок $M$ передается циклически. Далее каждое ядро будет полностью повторять действия однопроцессорной версии, только с уменьшенной матрицей $L_i$. Каждое ядро получает некоторые строки итогового произведения, соответствующие высланным ему столбцам матрицы $L$, которые оно записывает в матрицу-произведение. В итоге необходимо $\frac{s \times N}{nit} + s \times N$ пересылок машинных слов, $\frac{3 \times s^2 \times N \times n}{nit \times (\log_2 N - \log_2 \log_2 N)}$ операций и $\frac{s^2 \times s}{nit}$ перезаписей машинных слов на формирование результата. Эффективность распараллеливания составляет около 70%. Это обусловлено тем, что есть не зависящее от $nit$ слагаемое.

Второй вариант отличается от первого тем, что оба столбца делятся на части, и каждое ядро получает соответствующую пару блоков (часть столбцов $L$ и часть столбцов $M$), которые он перемножает, получая соответствующие части матрицы-произведения. При такой реализации необходимо разложить число ядер $nit$ на множители $nit_1$ и $nit_2$, первый столбец будем делить на $nit_1$ частей, а второй — на $nit_2$. Тогда необходимо $\frac{s \times N}{nit_1} + \frac{s \times N}{nit_2}$ перезаписей машинных слов, $\frac{3 \times s^2 \times N \times n}{(nit_1 \times nit_2 \times (\log_2 N - \log_2 \log_2 N))}$ операций, и затем $\frac{s^2 \times n}{nit_1 \times nit_2}$ перезаписей для сбора результатов. Выгодно брать $nit_1 \approx nit_2$. Результаты оказались примерно такими же, ускорение составило около 75%.

Третий вариант программы рассчитан на вычислительные узлы с разделенной памятью. У каждого узла хранится соответствующая ему часть первого столбца, и часть второго столбца, каждая часть примерно $\frac{N}{\sqrt{p}}$ строк, где $p$ — количество вычислительных узлов. Далее на каждом вычислительном узле происходит распараллеливание между ядрами по первому варианту. В итоге после умножения каждый вычислительный узел будет владеть матри-

цей размера $(s \times n) \times s$. Эти матрицы необходимо сложить, чтобы получить матрицу-произведение. Сложение будет происходить циклически, каждый столбец складывается по очереди. Оценим сложность такого алгоритма. Необходимо $\frac{3 \times s^2 \times N \times n}{p \times nit \times (\log_2 N - \log_2 \log_2 N)}$ операций и $\frac{2 \times (p-1) \times s^2 \times n}{p}$ пересылок на сбор информации. При $p$ меньше 20 достигается примерно 70% ускорение по $nit$ и 85% ускорение по $p$. Однако, при больших значениях $p$ (около 50–60) ускорение уменьшается, и составляет примерно 60% в зависимости от количества вычислительных узлов.

Итак, использование описанных алгоритмов позволяет вычислить произведение матриц в 32 раза быстрее при непараллельной версии программы (на оборудовании СКИФ МГУ). Ускорение же при распараллеливании составляет около 200 раз для умножения матриц машинных слов и примерно 400 раз при вычислении скалярного произведения столбцов.

### Список литературы

[1] Coppersmith D. Solving homogeneous linear equations over GF(2) via Block Wiedemann Algorithm // Math. of Comp. — 1994. Vol. 62. No. 205. — P. 333–350.

[2] Богачев К. Ю. Основы параллельного программирования. — Бином. Лаборатория знаний, 2003.