УДК 004.415.2.031.43 DOI 10.18522/2311-3103-2024-5-48-58

И.И. Левин, Е.А. Дудников

СТРУКТУРНАЯ МОДИФИКАЦИЯ МЕТОДА ХАФФМАНА ДЛЯ СЖАТИЯ ПЛОТНЫХ ПОТОКОВ ДАННЫХ БЕЗ ПОТЕРЬ НА РВС

Современные запросы общества требуют решения целого ряда вычислительно трудоемких задач в режиме реального времени. Для подобных решений необходимы огромные вычислительные мощности, широкополосные высокоскоростные каналы передачи данных и внушительные объемы памяти. Обеспечить подобные запросы можно за счет разработки и внедрения новых технологий, наращивания технической инфраструктуры, что потребует значительных финансовых и временных затрат. Облегчить подобный переход, используя существующую техническую базу, можно за счет использования алгоритмов сжатия данных в реальном времени. Средства сжатия данных в темпе поступления могут повысить скорость вычислений, передачи данных, снизить занимаемый объем при хранении, используя имеющуюся инфраструктуру. Современные технические платформы на базе CPU не способны обеспечить потоковую обработку данных в темпе их поступления, реальная производительность подобных систем не превышает 10 % от пиковой. Новой платформой для систем сжатия данных без потерь в темпе поступления могут стать реконфигурируемые вычислительные системы (РВС) на базе программируемых логических интегральных схем (ПЛИС). Однако для эффективной работы подобных систем требуется разработка новых методов с применением структурных вычислений, позволяющих полностью раскрыть потенциал ресурса ПЛИС. В данной работе представляется реализация на РВС модификации динамического алгоритма кодирования Хаффмана, которая позволяет создавать префиксные коды оптимальной длины и обрабатывать плотные потоки данных в темпе поступления с пропускной способностью не менее 128 Гбит\с. Производительность разработанной модификации в 5 раз превосходит наилучшую известную комплементарную реализацию на базе FPGA на один вычислительный конвейер.

РВС; динамическое кодирование Хаффмана; обработка данных в реальном времени.

I.I. Levin, E.A. Dudnikov

STRUCTURAL MODIFICATION OF THE HUFFMAN METHOD FOR COMPRESSION OF DENSE DATA STREAMS WITHOUT LOSS ON A RCS

Modern society demands require solving a whole range of computationally intensive tasks in real time. Such solutions require enormous computing power, broadband high-speed data transmission channels and impressive memory capacities. Such demands can be met by developing and implementing new technologies, expanding the technical infrastructure, which will require significant financial and time costs. Such a transition can be facilitated using the existing technical base by using real-time data compression algorithms. Data compression tools at the rate of receipt can increase the speed of calculations, data transfer, and reduce the occupied space during storage, using the existing infrastructure. Modern CPU-based technical platforms are not capable of providing streaming data processing at the rate of their receipt; the actual performance of such systems does not exceed 10% of the peak. Reconfigurable computing systems (RCS) based on programmable logic integrated circuits (FPGAs) can become a new platform for lossless data compression systems at the rate of receipt. However, for the efficient operation of such systems, it is necessary to develop new methods using structural calculations that allow the full potential of the FPGA resource to be unleashed. This paper presents the implementation of a modification of the dynamic Huffman coding algorithm on the RCS, which allows creating prefix codes of optimal length and processing dense data flows at the rate of receipt with a throughput of at least 128 Gbit/s. The performance of the developed modification is 5 times higher than the best known complementary implementation based on FPGA per computing pipeline.

RCS; dynamic Huffman coding; real-time data processing.

Введение. Современные требования к качеству решения вычислительно трудоёмких научно-технических задач неуклонно ведёт к увеличению объёмов производимой и обрабатываемой информации. Так по оценкам специалистов человечеством производится порядка 320 миллионов терабайт данных ежедневно, а общий объём произведённой информации к 2025 году превысит 175 зеттабайт [1]. Согласно этим отчётам объем репли-

цированной информации в девять раз выше объема оригинальной произведенной информации, и подобное смещение будет только нарастать. Большая часть скопированной и использованной информации носит временный характер и зачастую служит для одноразового использования. Растёт объём метаданных и данных различных сервисов о конкретных пользователях, который будет значительно превышать объём данных, созданный этими пользователями.

Немаловажную роль в подобном бурном росте сыграла разразившаяся в 2019 году пандемия, потребовавшая перехода множества организаций, связанных с производством и образованием, на удалённые формы взаимодействия. Основными отраслями, генерирующими информацию, по-прежнему остаются промышленность, здравоохранение, финансовый сектор, развлечения и медиабизнесс [2]. Благодаря снижению стоимости пользования облачными хранилищами происходит замена физических серверов. Уже сегодня около 94% крупных компаний используют облачные хранилища, а в 2025 году туда может переместиться до 80% всей рабочей нагрузки в мире [3]. Всё это ведёт к необходимости хранения, передачи и обработки огромного объема данных в том числе и в режиме «реального времени». К отраслям, требующим обработки данных в реальном времени, можно отнести военную, космическую области, задачи ядерной физики, радиолокации, геолокации, предиктивной аналитики, промышленные автоматические системы управления производством, энергетику, трансляции потоков аудиовизуальной информации от источника к конечному пользователю (мультимедиа, телекоммуникации) [4].

Непременными требованиями к задачам реального времени являются не только корректность вычислений, но и временные рамки. Латентность системы может быть различной, но она должна быть гарантированной и предсказуемой - нарушение сроков исполнения задачи расценивается как возможный отказ системы. Принадлежность системы к классу задач реального времени никак не связана с ее быстродействием: время обработки может быть различным, требования сроков исполнения задачи определяется техническим заданием и логикой функционирования системы [5].

Вычислительно трудоёмкие задачи реального времени и растущие объёмы данных требуют увеличения мощности вычислительных систем, объемов общей и оперативной памяти, увеличения пропускной способности каналов передачи данных. Для этого необходимо перестроить значительную часть существующей инфраструктуры. Это влечёт за собой существенные финансовые и временные затраты на разработку и внедрение новых технологий обработки, передачи и хранения информации. Облегчить подобный переход могут новые высокоскоростные системы сжатия данных, позволяющие более эффективно использовать уже существующую техническую базу. Уже сейчас крупные компании внедряют новые алгоритмы сжатия данных для облачных хранилищ [6]. Одним из основных требований к подобным системам является возможность восстановления исходных данных без потерь.

Для решения трудоёмких задач выделяют основные типы вычислительных устройств: центральные процессоры (CPU), графические процессоры (GPU) специализированные устройства (ASIC) и программируемые вентильные матрицы (FPGA). Эти устройства имеют между собой явные различия. Большинство задач так или иначе можно выполнять с помощью CPU, однако другие типы процессоров позволяют решать многие вычислительно сложные задачи быстрее с меньшими энергозатратами.

По причине широкого распространения большая часть задач по сжатию данных попрежнему выполняется системами на базе CPU. Однако подобные системы не способны обеспечить требуемый уровень производительности. Исследования показывают, что при сжатии данных алгоритмами семейства DEFLATE производительность ядра составляет от нескольких десятков до нескольких сотен Мбит/с. При использовании специализированных многоядерных систем независимо от типа многоядерной машины и числа ядер скорость обработки потока составляет от 512 Мбит/с до 3,4 Гбит/с в зависимости от коэффициента сжатия [7]. Наращивание аппаратного ресурса не приводит к линейному

росту производительности, что связано с проблемами распараллеливания процессов, задействованности ядер и межпроцессорными связями. Причина кроется в самой структуре универсальных процессоров, которые не предназначены для высокопроизводительных систем с конвейерной организацией вычислений.

Эффективность GPU при переносе на них классических методов сжатия без потерь показывает сопоставимые с CPU результаты 2,5–7,3 Гбит/с. Применение GPU позволяет повысить производительность за счёт вычислительных мощностей, но это требует разработки и адаптации существующих методов сжатия. Использование специализированных адаптированных методов на основе существующих, предварительная обработка данных (например, преобразование Барроуза-Уилера) [8] позволяет повысить производительность GPU, но подобный рост также может приводить к кратному снижению степени сжатия и росту объёма вычислений.

Технической основой для устройств по сжатию данных могут быть специализированные устройства на базе заказных кристаллов или реконфигурируемые вычислительные системы (РВС) на базе FPGA. Производительность коммерческих реализаций методов сжатия данных без потерь на базе ASIC колеблется в пределах 20 Гбит/с на кристалл [9]. Разработка подобных систем является долгим и затратным предприятием. Более того, при внесении изменений в алгоритм, обнаружении ошибки, модернизации потребуется повторная разработка и перевыпуск всей серии устройств. Производительность комплементарных систем на базе CPU и FPGA не превышают 120,4 Гбит/с на один кристалл. Наилучшей реальной производительности одного вычислительного конвейера в 20 Гбит/с добились специалисты CDSC и Xilinx [10]. Добиться подобной производительности разработчикам удалось за счет применения статических энтропийных методов. Подобные решения эффективны лишь для заранее известных типов данных, близких по содержанию к эталону. При обработке разнородных данных без использования подготовленных статистических таблиц такое решение приведет к значительному снижению степени сжатия входного сообщения.

Достичь оптимального уровня сжатия для алгоритмов кодирования данных в темпе их поступления на РВС с сохранением высокой пропускной способности можно с применением принципов структурных вычислений. Однако традиционные методы архивирования не учитывают специфику работы РВС, поэтому для эффективной реализации алгоритмов сжатия данных необходимо разработать новые методы управления потоками данных и организации вычислений. Применение подобных методов позволит превзойти производительность уже существующих систем сжатия данных без потерь на РВС в темпе поступления и достичь производительности 128 Гбит/с и более на один конвейер. Предполагается, что разработка и внедрение новых методов позволит сократить задействованный аппаратный ресурс и сократить латентность вычислений по сравнению с известными методами. На основе вышеизложенного можно сделать заключение, что разработка новых методов эффективного сжатия высокоскоростных потоков данных в темпе их поступления на РВС является актуальной задачей.

Постановка задачи. Современные системы сжатия данных без потерь (Deflate, Snappy, Gipfeli, Zopfli, Brotli и т.д.) являются комплексными, включают несколько различных алгоритмов преобразования и кодирования. Важным элементом подобных систем являются энтропийные алгоритмы, позволяющие генерировать префиксные коды оптимальной длины. Энтропийное кодирование основано на частоте повторения символов в сообщении и направленно на устранение избыточности и уменьшение занимаемого сообщением объема.

Одним из наиболее популярных энтропийных методов, широко используемым во многих архиваторах и протоколах данных, является разработанный в 1952 году метод Хаффмана [11]. Идея метода заключается в том, что символу алфавита сообщения $s_i \in \{s_1, s_2, \dots, s_m\}$ с фиксированной длинной и соответствующей частотой появления $w_i \in \{w_1, w_2, \dots, w_m\}$, $w_1 \le w_2 \le \dots \le w_m$ присваивается новый код переменной длины l_i, зависящей от w_i так, что чем выше частота появления, тем короче новый код символа. Тогда минимизированную длину нового сообщения можно представить как выра-

жение $\sum_{i=1}^{m} w_i l_i$. являющимся производным от уравнения средней длины кода Шеннона [12]. Полученное сообщение может быть однозначно декодировано благодаря префиксности кода.

Существующие алгоритмы кодирования Хаффмана можно разделить на статические, динамические и адаптивные. Динамический метод требует двух вычиток передаваемого сообщения из памяти: первый раз для сбора статистики и генерации кодов, второй раз для кодирования. Этот алгоритм обеспечивает оптимальную степень сжатия кодируемого сообщения. В случае, когда можно заранее спрогнозировать распределение вероятностей появления символов в сообщении, обосновано применение статического метода Хаффмана с применением заранее рассчитанных таблиц кодов. Такой подход не позволяет создавать коды оптимальной длины, что снижает степень сжатия. Степень сжатия снижается еще сильнее, если обрабатываются разнородные или неизвестные заранее данные. Для некоторых сфер, требующих обработки данных в темпе их поступления, таких, например, как телекоммуникационные системы, статический метод Хаффмана может оказаться малоэффективным. Для подобных задач применяют адаптивные варианты алгоритма Хаффмана. Адаптивный метод позволяет решать задачу за одно прочтение без использования таблиц статистики. Однако по сравнению с динамическим, адаптивный алгоритм требует выполнения большего числа операций, и обладает более низким коэффициентом сжатия данных. Исходя из соотношения степени сжатия к сложности вычислений, можно сделать вывод, что при обработке больших массивов разнородных данных динамический алгоритм Хаффмана является более эффективным по сравнению с адаптивным или статическим алгоритмом. Однако необходимость двойного чтения сообщения из памяти не позволяет кодировать плотные потоки данных на традиционных вычислительных системах в режиме реального времени.

Известен ряд модификаций динамического алгоритма Хаффмана, предлагающих построение префиксных кодов с минимальной избыточностью (Minimum-Redundancy Codes), направленных на снижение вычислительной сложности и уменьшение аппаратных затрат. Например, в работах [13, 14] предлагается параллельно заполнять многомерный массив дерева Хаффмана и рассчитывать длины кодов с последующей генерацией кодов за линейное время. В работе [15] предложено построение дерева Хаффмана без создания и хранения таблицы массива данных, описывающих дерево и прохода по этой таблице при генерации новых кодов, а путем расчета длин кодов и последующей генерацией кодов. В зависимости от используемого метода полученные коды могут отличаться, но уровень энтропии новых кодов и минимальная избыточность полученных кодов сохраняется для каждого символа алфавита согласно неравенству Крафта $K = \sum_{i=1}^{m} 2^{-l_i} \leq 1$, где l_i это число бит в коде символа (длина кода) [16]. Тем не менее, подобные модификации не избавляют от недостатков динамического алгоритма Хаффмана, а именно необходимости двойного чтения кодируемого сообщения.

Структурная реализация алгоритма Хаффмана на РВС позволит избавиться от подобных проблем. В частности, блок буферизации и синхронизации входного сообщения позволит реализовать динамический алгоритм Хаффмана на РВС, выполняя лишь одну загрузку сообщения из внешнего источника. Оценив максимальную пропускную способность известных реализаций на базе FPGA, были выдвинуты технические требования для разрабатываемой модификации. Структура должна обрабатывать плотные потоки данных на скорости 128 Гбит/с. Для соответствия подобным требованиям при ширине шины данных 512 бит рабочая частота системы должна быть не ниже 250 МГц. При подобном темпе поступления и размере пакета 64 Кбайт (максимальный размер IP-датаграммы) для обработки сообщения потребуется 1024 такта, что и будет являться интенсивностью (скважностью) системы. Соответственно, статистика появления символов в сообщении для каждого блока должна формироваться один раз в 1024 такта. Использование известных последовательных алгоритмов не позволит должным образом распараллелить задачу. Кроме того, задача кодирования методом Хаффмана относится к классу задач с переменной интенсивностью потоков данных и последовательная обработка или вызывает

нерегулярные задержки, или увеличивает общую латентность решения до максимальной. При этом система буферизации и синхронизации, как для классического динамического алгоритма Хаффмана, так и его модификаций может занять значительный аппаратный ресурс, который может превосходить затраты вычислительной части алгоритма. Глубина подобной системы непосредственно зависит от латентности блока расчета и построения таблицы новых кодов, поэтому целесообразно использовать модификацию, обеспечивающую наибольшую скорость исполнения.

Процесс получения новых кодов можно представить как заполнение двумерной матрицы $A_{(n,m)}$, где m – это количество строк, соответствующее числу символов алфавита, имеющих вес, а n - наибольшая ширина машинного слова (длина кода). Было рассчитано, что для блока сообщения в 65536 символов наибольшая длина кода n_{max} =22 при вырожденном дереве, отображающем последовательность Фибоначчи:

$$F_n = F_{n-1} + F_{n-2},$$

где $F_0, F_1, F_3 = 1$ и $F_n \leq 65536$.

Для сокращения вычислительной нагрузки некоторые алгоритмы ограничивают длину кода. Согласно уравнению средняя длина кода $K = \log_2 w_{max}$, где w_{max} – максимально допустимый вес символа. При $w_{max} = 65636$ средняя длина кода составит $K = 16$, что совпадает с регламентами некоторых технических протоколов [17].

Описание модифицированного метода Хаффмана с разработанными блоками построения дерева и расчета новых кодов. Классический метод Хаффмана предполагает заполнение таблицы узлов бинарного дерева с последующим вычислением кодов путём либо прямого прохода от вершины к корню, либо рекурсивного прохода со ссылками на предыдущие узлы. Сложность динамического алгоритма Хаффмана в значительной степени обуславливается числом обращений к многомерному массиву предварительно построенного дерева, хранение которого требует значительных аппаратных затрат. Максимальное число обращений к узлам таблицы кодов определяет сложность алгоритма и формирует минимальную латентность процесса генерации кодов S_{lat}. При моделировании классического метода реальная латентность блоков построения дерева Хаффмана и создания новых кодов составила 5734 такта. Для эффективной работы исходный граф был векторизирован [18] с целью соответствия техническим требованиям задачи кодировки данных в темпе их поступления. Блоки сбора статистики и сортировки обрабатывают входное сообщение за 1024 такта и формируют скважность поступления пакетов S_p. Для успешной работы необходимо, чтобы скважность работы структуры G_{code} отвечала условию $S_{code} \leq S_p$. В полученном векторизованном графе скважность работы блока будет равна его латентности $S_{code} = S_{lat}$. Если $S_{code} > S_p$, то уменьшить скважность системы построения кодов можно за счёт распараллеливания на верхнем иерархическом уровне путем параллельного подключения нескольких макроконвейеров [19] G_{code}. Реализация классического метода Хаффмана потребует параллельной установки пяти макроконвейеров (рис. 1).

Было рассчитано, что распараллеливание на нижнем (итерационном) иерархическом уровне занимает меньший ресурс по сравнению с распараллеливанием по верхнему уровню с сохранением функциональности. При сохранении условия $S_{code} \leq S_p$ суммарный аппаратный ресурс будет сокращён до минимума. Анализ метода, описанного работе [20], позволил определить, что распараллеливание по каналам доступа к памяти позволяет уменьшать латентность вычислительной структуры вместе с увеличением интенсивности обработки информации. Было выполнено распараллеливание вычислительной структуры классического алгоритма построения дерева Хаффмана по каналам доступа к памяти родителей, направленное на уменьшение её латентности, что позволило существенно уменьшить аппаратные затраты на реализацию подсистемы синхронизации.

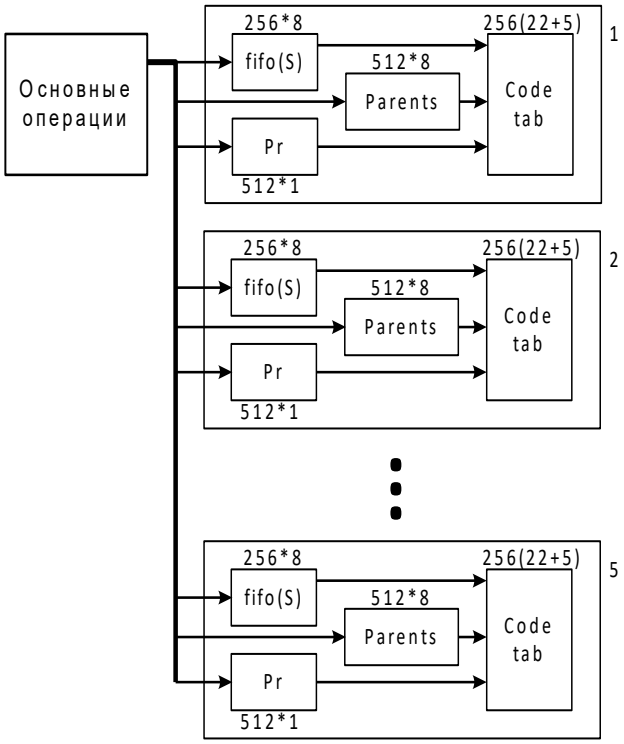

Рис. 1. Макроконвейерная реализация вычислительной структуры при реализации классического алгоритма построения дерева Хаффмана на РВС

Моделирование модификации классического метода Хаффмана показало, что применение параллельного доступа к памяти позволило снизить латентность вычислительной структуры S_{code} до 1792 тактов. Поскольку S_{code} превышает скважность поступающих данных, то для организации обработки входящего сообщения в темпе поступления требуется установка двух параллельных вычислительных блоков на макроуровне. Дальнейшее снижение латентности вычислительного блока потребовало разработки новой модификации динамического алгоритма.

В новой разработанной модификации Хаффмана, информационный граф которой представлен на рис. 2, процессы построения дерева и вычисления кодов выполняются параллельно. Подобный подход позволяет получать коды без предварительного построения дерева, заполнения и хранения таблицы узлов с ссылками на родителей, а также многократного прохода по некоторым ветвям при генерации кода для каждого символа алфавита.

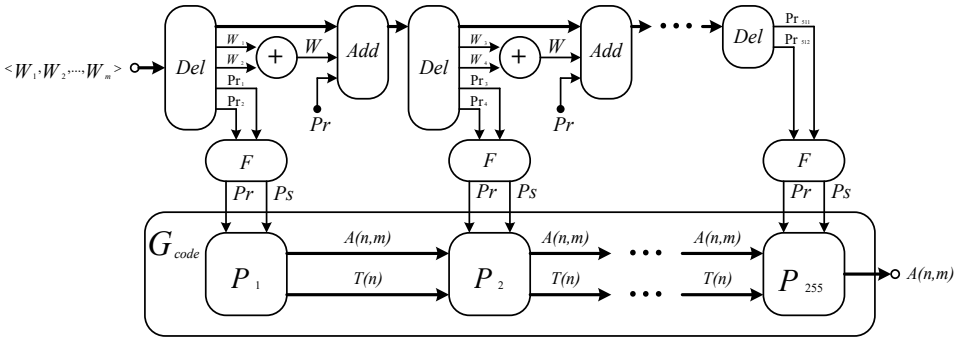

Рис. 2. Информационный граф разработанной модификации метода Хаффмана с распараллеливанием по каналам доступа к памяти

В основе новой модификации лежит понимание того, что дерево Хаффмана можно представить в виде набора уровней, а информации о числе уровней дерева и их наполнении достаточно для построения новых кодов и расчета их длин. Основу дерева представляют элементы, родителем которых является корневая вершина дерева. Верхний уровень дерева составляют листья с наименьшим весом. Любое подобное дерево может мыть описано совокупностью элементов на каждом из уровней. Важным изменением при распараллеливании по числу каналов доступа к памяти является преобразование информационного графа $G_{code} = \bigcup_{j=1}^{k} P_j$, где $k = (m-1)$, и подграфа P для обработки не одного, а пары символов. Операции Del последовательно удаляют из массива весов $< W_1, W_2, \dots, W_m >$ два элемента. Веса удалённых элементов складываются, сумма получает признак и как новый элемент помещается в массив операцией Add. Признак каждого элемента поступает в подграф P и обрабатывается подграфами L, располагающиеся внутри графа P в виде равномерно расширяющегося к низу дерева.

Это нововведение позволяет одновременно писать признаки данных по двум выходящим веткам на каждом уровне дерева, что сокращает общее время построения кодов. В зависимости от того, чем являются удаляемые элементы, в блок построения дерева передается признак $P(j)$:

- при $P(j) = 2$ – оба удаляемых элемента являются символами;
- при $P(j) = 1$ – один элемент является символом, другой – узлом;
- при $P(j) = 0$ – оба удаляемых элемента являются узлами.

В блоке построения дерева, показанном на рис. 3, представлены новые разработанные блоки, которые рассчитывают общее число потомков-символов и потомков-узлов после каждого родителя, расположенных уровнем выше. Эти суммы признаков символов $C(j)$ и узлов $U(j)$, полученные на каждой итерации алгоритма, а также исходные коды символов $W_1 \div W_m$, расположенные в порядке увеличения их весов накапливаются в памяти $BUF1$.

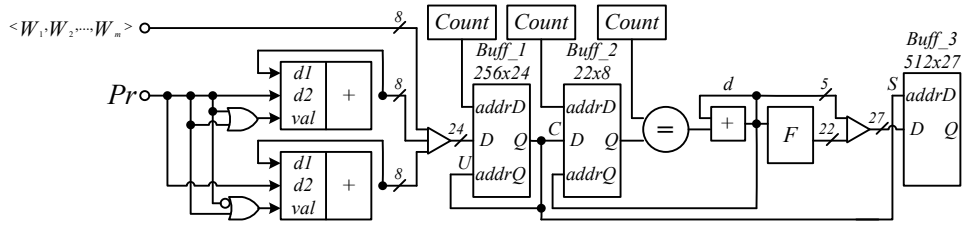

Рис. 3. Реализация на РВС вычислительной структуры блока построения дерева Хаффмана

После подсчета всех признаков элементов дерева определяется число символов, расположенных ниже каждого из родителей $A(i)$, где $i \le 22$ – предельное количество уровней в дереве Хаффмана для сообщения 64Кбайт. Это значение накапливается в памяти $BUF2$. Данный подход организован с применением двух очередей в памяти FIFO. Первая – это очередь признаков, указывающая номера строк для записи, которую также помещают в очередь и достают оттуда при получении последующих признаков со значением «0» для следующего столбца. Вторая очередь представляет собой производные от позиции Ps данные, из которых формируется новый код. Поскольку дерево Хаффмана обладает строгой вертикальной иерархией от корневого узла к листьям, то на конкретных этапах j сумма признаков символов $C(j)$ определяет общее число символов, расположенных ниже родителей на каждом из уровней. Количество узлов $U(j)$, расположенных ниже родителей на каждом из уровней, определяет операции, которые необходимо провести на каждом этапе j. На основе полученных данных $A(i)$ в блоке F рассчитывается новый код e с соответствующей длиной кода d и записывается в таблицу перекодировки $BUF3$. позиция символа в отсортированном массиве данных в зависимости от его веса. Оценка

вычислительной трудоёмкости при моделировании разработанного алгоритма показала, что латентность полученной модификации $S_{code} = 1024$ и не превышает скважность поступления данных S_p.

В процессе анализа были синтезированы и промоделированы несколько модификаций динамического алгоритма Хаффмана: «классический» алгоритм, модификация Хашемиана, модификация Алексеева, соответствующая по вычислительной сложности модификации Моффата, а также новый разработанный алгоритм. Все модификации, реализованные на ПЛИС XC7K355T фирмы Xilinx серии Kintex 7, обеспечили кодирование плотного потока входящих данных со скоростью 128 Гбит/с и частотой работы кристалла 250 МГц. Сравнение характеристик различных модификаций представлены в табл. 1.
P(lut), P(bram) – характеристики, показывающие долю аппаратного ресурса, занимаемую вычислительной структурой рассмотренных алгоритмов построения дерева Хаффмана, относительно ресурса, занимаемого классическим алгоритмом.

В табл. 2 представлены аппаратные затраты, используемые на реализацию рассмотренных алгоритмов динамического алгоритма кодирования Хаффмана с учётом блоков подсчёта вероятностей и сортировки, по таким параметрам как количество используемых логических таблиц (LUT), количество триггеров (Flip-Flops) и количество блоков встроенной памяти ПЛИС объемом 36Кб (BRAM).

Таблица 1

Сравнение различных реализаций блоков построения дерева Хаффмана и генерации кодов

параметр	latency	LUT	BRAM, huff	BRAM, sync	BRAM, total	P(lut)	P(bram)
Классический	5734	1051	20	171	191	1	1
Алексеев\Моффат	3968	3476	2	114	116	3.31	0.61
Модифицир. классический	1792	1124	18	57	75	1.07	0.4
Хашемиан	1280	199	3	44	47	0.2	0.25
Разработанный	1024	177	3	29	32	0.17	0.17

Таблица 2

Латентность и аппаратные затраты на реализацию рассматриваемых модификаций динамического алгоритма Хаффмана

параметр	latency	LUT	BRAM,36K	Flip-Flops
Классический	5734	50859	170,5	81375
Алексеев\Моффат	3968	53284	133	87919
Хашемиан	1280	50007	98,5	80011
Разработанный	1024	49985	90,5	74977

Реализация любого из рассмотренных алгоритмов позволяет достигать необходимого уровня производительности вычислительной структуры для решения поставленной задачи, однако анализ полученных результатов показал, что вычислительная структура разработанной модификации имеет наименьшую латентность (latency) и требует меньше аппаратных затрат по сравнению с другими модификациями. При масштабировании вычислительной структуры производительность РВС будет увеличиваться линейно, что позволит при необходимости выполнять кодирование потоков данных на большей скорости.

Заключение. Представленная в данной статье разработанная модификация построения дерева Хаффмана и расчёта новых кодов позволяет эффективно реализовать динамический алгоритм кодирования Хаффмана на РВС. Согласно парадигме структурных вычислений данная реализация позволяет убрать нелинейность структуры за счёт

разбиения на итерационные уровни и добиться высокого уровня распараллеливания задачи. Снижение вычислительной сложности и латентности вычислительной структуры позволило при реализации на ПЛИС повысить удельную производительность с сохранением заданной скорости поступления потока данных. Более того, новые коды рассчитываются динамически, что позволяет получать новые коды оптимальной длины для текущего сообщения и обеспечивает максимальную степень сжатия. За счет системы буферизации и синхронизации достаточно вычитывать входное сообщение лишь один раз, что снижает межпроцессорные обмены. Предложенное решение позволяет обрабатывать высокоскоростные плотные потоки данных в темпе поступления.

Анализ аппаратных затрат и вычислительной сложности показал, что снижение вычислительной сложности и латентности в разработанной модификации позволило уменьшить занимаемый вычислительным блоком аппаратный ресурс, сократить объем используемой внутренней памяти ПЛИС на систему буферизации и отказаться от распараллеливания на макроуровне.

Было выполнено функциональное моделирование предложенного решения. Производительность новой модификации в 5 раз превосходит лучшую известную реализацию на PBC на одно вычислительное ядро с рабочей частотой в 1,25 раза выше, с сохранением оптимальной степени сжатия методом Хаффмана для текущего сообщения.

Разработанная модификация позволяет организовать сжатие плотного потока данных на скорости до 128 Гбит/с, что обеспечивает потребности современных интерфейсов, таких как PCI Express 3.0, Thunderbolt 3, Ethernet 100GbE, InfiniBand EDR 4X. При этом масштабирование вычислительного блока позволит обрабатывать потоки данных на большей скорости.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Berisha B., Mëziu E., Shabani I.* Big data analytics in Cloud computing: an overview // Journal of Cloud Computing. – 2022. – No. 11 (1).
2. *Morgan S.* The 2020 Data Attack Surface Report, White Paper. – https://cybersecurityventures.com/wp-content/uploads/2020/12/ArcserveDataReport2020.pdf.
3. 2020: Oracle’s Top 10 Cloud Predictions. – https://www.oracle.com/a/ocom/docs/cloud/oracle-cloud-predictions-2020.pdf 2020г.
4. *Попов А.В.* Алгоритмы энтропийного кодирования при сжатии спектра телевизионного сигнала // T-Comm — Телекоммуникации и Транспорт. – 2013. – № 4. – С. 43-46.
5. *Луканов А.С.* Системы реального времени: учеб. пособие. – Самара: Изд-во Самарского университета, 2020. – 156 с.
6. https://aws.amazon.com/ru/blogs/aws/data-compression-improvements-in-amazon-redshift/.
7. *Promberger L., Schwemmer R., Fröning H*. Characterization of data compression across CPU platforms and accelerators // Concurrency and Computation: Practice and Experience. – 2021. – Vol. 35, No. 20 (e6465).
8. *Knorr F., Thoman P., Fahringer T.* ndzip-gpu: Efficient Lossless Compression of Scientific Floating-Point Data on GPUs // SC '21: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis Article. – Nov. 2021. – No. 93. – P. 1-14.
9. *Deaton J.D., Purdy J., Bacon* A. Smashing Big Data Costs with GZIP Hardware, AHA Products Group, Technical White Paper.
10. *Qiao W., Du J., Fang Z., Lo M. et. al.* High-Throughput Lossless Compression on Tightly Coupled CPU-FPGA Platforms // 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines. – 2018. – P. 37-44.
11. *Huffman D.A.* A method for the construction of minimum-redundancy codes // Proceedings of the IRE. – Sept. 1952. – Vol. 40, No. 9. – P. 1098-1101.
12. *Shannon C.E.* A mathematical theory of communication // Bell Sys. Tech. Jour. – 1948. – Vol. 27. – P. 398-403.
13. *Witten I.H., Moffat A., Bell T.C.* Managing Gigabytes: Compressing and Indexing Documents and Images. – 1st ed. – Van Nostrand Reinhold, New York, USA, 1994. – 429 p.
14. *Moffat A., Turpin A.* On the Implementation of Minimum Redundancy Prefix Codes // IEEE Transactions on Communications. – 1997. – Vol. 45, No. 10. – P. 1200-1207.
15. *Hashemian R.* Memory Efficient and High-speed Search Huffman Coding // IEEE Transactions on Communications. – 1995. – Vol. 43, No. 10. – P. 2576-2581.

16. *Kraft L.G*. A Device for Quantizing, Grouping, and Coding Amplitude-modulated Pulses. – Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1949.
17. International Standard ISO/IEC 10918-1:1993(E). CCITT Rec. T.81 (1992 E). Information technology — Digital compression and coding of continuous-tone still images — Requirements and Guidelines.
18. *Каляев А.В., Левин И.И.* Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. – М.: Янус-К, 2003. – 380 с.
19. *Каляев И.А., Левин И.И.* Реконфигурируемые вычислительные системы на основе ПЛИС. – Ростов-на-Дону: Изд-во ЮФУ, 2021. – 458 с.
20. *Alekseev K.N., Sorokin D.A., Levin I.I.* Structural-procedural implementation the huffman coding on reconfigurable computer systems in real time // Vestnik Komp'iuternykh i Informatsionnykh Tekhnologii (Herald of Computer and Information Technologies). – 2018. – No. 9. – P. 3-10.

REFERENCES

1. *Berisha B., Mëziu E., Shabani I.* Big data analytics in Cloud computing: an overview, *Journal of Cloud Computing,* 2022, No. 11 (1).
2. *Morgan S.* The 2020 Data Attack Surface Report, White Paper. Available at: https://cybersecurityventures.com/wp-content/uploads/2020/12/ArcserveDataReport2020.pdf.
3. 2020: Oracle's Top 10 Cloud Predictions. Available at: https://www.oracle.com/a/ocom/docs/cloud/oracle-cloud-predictions-2020.pdf 2020.
4. *Popov A.V.* Algoritmy entropiynogo kodirovaniya pri szhatii spektra televizionnogo signala [Entropy coding algorithms for spectrum compression of television signals], *T-Comm — Telekommunikatsii i Transport* [T-Comm — Telecommunications and Transport], 2013, No. 4, pp. 43-46.
5. *Lukanov A.S.* Sistemy real'nogo vremeni: ucheb. posobie [Real-time systems: a tutorial]. Samara: Izd-vo Samarskogo universiteta, 2020, 156 p.
6. Available at: https://aws.amazon.com/ru/blogs/aws/data-compression-improvements-in-amazon-redshift/.
7. *Promberger L., Schwemmer R., Fröning H*. Characterization of data compression across CPU platforms and accelerators, *Concurrency and Computation: Practice and Experience,* 2021, Vol. 35, No. 20 (e6465).
8. *Knorr F., Thoman P., Fahringer T.* ndzip-gpu: Efficient Lossless Compression of Scientific Floating-Point Data on GPUs, *SC '21: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis Article,* Nov. 2021, No. 93, pp. 1-14.
9. *Deaton J.D., Purdy J., Bacon* A. Smashing Big Data Costs with GZIP Hardware, AHA Products Group, Technical White Paper.
10. *Qiao W., Du J., Fang Z., Lo M. et. al.* High-Throughput Lossless Compression on Tightly Coupled CPU-FPGA Platforms, *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines,* 2018, pp. 37-44.
11. *Huffman D.A.* A method for the construction of minimum-redundancy codes, *Proceedings of the IRE,* Sept. 1952, Vol. 40, No. 9, pp. 1098-1101.
12. *Shannon C.E.* A mathematical theory of communication, *Bell Sys. Tech. Jour*., 1948, Vol. 27, pp. 398-403.
13. *Witten I.H., Moffat A., Bell T.C.* Managing Gigabytes: Compressing and Indexing Documents and Images. 1st ed. Van Nostrand Reinhold, New York, USA, 1994, 429 p.
14. *Moffat A., Turpin A.* On the Implementation of Minimum Redundancy Prefix Codes, *IEEE Transactions on Communications,* 1997, Vol. 45, No. 10, pp. 1200-1207.
15. *Hashemian R.* Memory Efficient and High-speed Search Huffman Coding, *IEEE Transactions on Communications,* 1995, Vol. 43, No. 10, pp. 2576-2581.
16. *Kraft L.G*. A Device for Quantizing, Grouping, and Coding Amplitude-modulated Pulses. Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1949.
17. International Standard ISO/IEC 10918-1:1993(E). CCITT Rec. T.81 (1992 E). Information technology — Digital compression and coding of continuous-tone still images — Requirements and Guidelines.
18. *Kalyaev A.V., Levin I.I.* Modul'no-narashchivaemye mnogoprotsessornye sistemy so strukturno-protsedurnoy organizatsiey vychisleniy [Modular-scalable multiprocessor systems with structural-procedural organization of computations]. Moscow: Yanus-K, 2003, 380 p.
19. *Kalyaev I.A., Levin I.I.* Rekonfiguriruemye vychislitel'nye sistemy na osnove PLIS [Reconfigurable computing systems based on FPGAs]. Rostov-on-Don: Izd-vo YuFU, 2021, 458 p.
20. *Alekseev K.N., Sorokin D.A., Levin I.I.* Structural-procedural implementation the huffman coding on reconfigurable computer systems in real time, *Vestnik Komp'iuternykh i Informatsionnykh Tekhnologii (Herald of Computer and Information Technologies),* 2018, No. 9, pp. 3-10.

Статью рекомендовал к опубликованию д.т.н., профессор Ю.А. Кравченко.

Левин Илья Израилевич – Южный федеральный университет; e-mail: levin@superevm.ru; г. Таганрог, Россия; тел.: +78634612111; кафедра интеллектуальных и многопроцессорных систем; зав. кафедрой; д.т.н.; профессор.

Дудников Евгений Александрович – e-mail: everlast-83@mail.ru; тел.: +79185914907; кафедра интеллектуальных и многопроцессорных систем; аспирант.

Levin Ilya Izrailevich – Southern Federal University; e-mail: levin@superevm.ru; Taganrog, Russia; phone: +78634612111; the Department of Intellectual and Multiprocessor Systems; head of the department; dr. of eng. sc.; professor.

Dudnikov Evgeny Alexandrovich – e-mail: everlast-83@mail.ru; phone: +79185914907; the Department of Intellectual and Multiprocessor Systems; graduate student.

УДК 004.056 DOI 10.18522/2311-3103-2024-5-58-68

С.Ю. Мельников, Р.В. Мещеряков, В.А. Пересыпкин

НЕКОТОРЫЕ АСПЕКТЫ ПРИМЕНЕНИЯ ТЕХНОЛОГИЙ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ЗАДАЧАХ ЗАЩИТЫ ИНФОРМАЦИИ (ОБЗОР)

Технологии искусственного интеллекта (ИИ) являются одной из наиболее динамично развивающихся областей обработки информации. Технологии ИИ используются как для обеспечения защиты информации, так и для организации атак на средства ее защиты. Сами системы ИИ могут содержать уязвимости и быть подвержены атакам различного рода. В статье анализируются некоторые аспекты применения технологий ИИ в задачах защиты информации. В рамках задачи биометрической идентификации рассматриваются угрозы подделки биометрических идентификационных признаков с целью получения прав доступа, и способы противодействия таким угрозам. Анализируются преимущества использования ИИ при защите информации в компьютерных системах и сетях по сравнению с традиционными средствами защиты. На примере акустического канала утечки информации от клавиатуры иллюстрируется использование технологий ИИ для обработки данных из технических каналов утечки. Рассматриваются методы повышения информативности таких каналов, использующие временные сверточные сети и модели классификации изображений, а также способы противодействия им. Отдельное внимание уделено вопросам информационной безопасности в набирающих популярность системах сжатия и передачи информации без значительных смысловых потерь (направление Semantic Communications). Рассматриваются ряд вопросов информационной безопасности, возникающих при использовании больших языковых моделей типа ChatGPT, способных массово генерировать уникальный «человекоподобный» контент и использовать его для организации фишинговых и других атак социальной инженерии. Описана атака на системы ИИ с использованием скрытого канала. Уделено внимание необходимости развития технологий доверенного искусственного интеллекта.

Информационная безопасность; кибербезопасность; технический канал утечки; искусственный интеллект; доверенный искусственный интеллект.

S.Yu. Melnikov, R.V. Meshcheryakov, V.A. Peresypkin

SOME ASPECTS OF APPLICATION OF ARTIFICIAL INTELLIGENCE TECHNOLOGIES IN INFORMATION SECURITY (REVIEW)

Artificial intelligence (AI) technologies are one of the most dynamically developing areas of information processing. AI technologies are used both to ensure the information security and to organize attacks on information security tools. AI systems themselves may contain vulnerabilities and be susceptible to various types of attacks. The article analyzes some aspects of the use of AI technologies in information security tasks. Within the framework of the task of biometric identification, threats of falsification of biometric identification characteristics in order to obtain access rights, and ways to counter such threats are considered. The advantages of using AI in protecting information in computer systems and networks in comparison with traditional means of protection are analyzed. Using the example of an acoustic channel of information leakage from a keyboard, the use of AI technologies for processing data from technical leakage channels is illustrated. Methods for increasing the information content of such channels using