ПРИМЕНЕНИЕ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ПРИ РАЗРАБОТКЕ АЛГОРИТМОВ СЖАТИЯ ДЛЯ СИСТЕМЫ ТРАНСЛЯЦИИ ТАБЛИЦ ПОТОКОВ СЕТЕВОГО ПРОЦЕССОРНОГО УСТРОЙСТВА

Н.И. Никифоров, Д.Ю. Волканов (Москва)

Введение

В настоящее время активно развиваются технологии программно-конфигурируемых сетей (ПКС) [1]. Для работы ПКС требуются высокопроизводительные коммутаторы, которые выполняют функцию передачи данных. Возникает задача разработки программируемого сетевого процессорного устройства (СПУ) [2], являющегося основным функциональным элементом коммутаторов. В работе рассматривается коммутатор, функционирующий под управлением протокола OpenFlow [3].

OpenFlow – один из наиболее распространённых протоколов для управления сетевым коммутатором ПКС. В этой статье рассматривается OpenFlow версии 1.3 [3]. Правила обработки пакетов, в котором представляются в виде таблиц потоков, которые могут быть как обычными, так и групповыми. В данной работе рассматриваются только не групповые таблицы потоков. В СПУ таблицы потоков представляются в виде программы обработки заголовков сетевых пакетов на языке ассемблера, получаемой с помощью транслятора таблиц потоков. Таблица потоков – это набор правил, определённых протоколом OpenFlow. Каждое правило содержит поле признака, битовую строку, по которой пакет может быть идентифицирован, и набор действий, которые СПУ выполняет с этим пакетом.

Данная работа посвящена разработке алгоритмов сжатия данных [4] для применения в трансляторе таблиц потоков в рамках рассматриваемой архитектуры сетевого процессорного устройства. Поскольку современные таблицы потоков занимают до нескольких десятков мегабайтов памяти [5], возникает задача сжатия таблиц потоков, для использования рассматриваемого СПУ в коммутаторах ПКС. Данная работа имеет следующую структуру: в части 1 приводится описание предметной области, в части 2 производится постановка задачи, в части 3 приводится описание имитационной модели СПУ, в части 4 описывается структура транслятора таблиц потоков, в части 5 описывается проведённое экспериментальное исследование на имитационной модели СПУ.

1. Описание предметной области

1.1 Архитектура сетевого процессора (RuNPU)

В СПУ используется конвейерная архитектура, каждый конвейер состоит из 10 вычислительных блоков. Вычислительный блок – это набор более низкоуровневых RISC ядер, которые в данной работе не рассматриваются. Каждый вычислительный блок имеет доступ к участку памяти, в котором располагаются микрокод и данные. Существует ограничение на количество тактов, которое один пакет может обрабатываться на вычислительном блоке, оно соответствует 25 тактам. Данное ограничение обусловлено требованием к производительности сетевого процессора, а именно, фиксированное время обработки одного пакета на сетевом процессоре. Также один вычислительный блок имеет доступ к 64 килобайтам памяти. Из-за особенностей микроархитектуры, отсутствует отдельная область памяти, в которой хранятся данные. Поэтому микрокод содержит в себе все данные, необходимые для классификации пакетов.

1.2 Язык ассемблера сетевого процессора

Для описания программ обработки сетевых пакетов в рассматриваемой архитектуре сетевого процессора используется язык ассемблера. Вычислительный блок конвейера содержит единственный регистр общего назначения длиной 128 бит (регистр-аккумулятор), который выступает в качестве регистра-операнда и/или регистра-результата команды. Память, доступ к которой осуществляется через команды, разделена на нескольких частей: область заголовка пакета; область метаданных (размер заголовка, номер порта, на который пришёл пакет, маска выходных портов, пользовательские метаданные и др.). Также вычислительный блок конвейера содержит регистр смещения. При выполнении команд, осуществляющих доступ в область памяти, используется адрес равный сумме адреса, прописанного в команде, и значения регистра смещения.

Язык ассемблера сетевого процессора предназначен для описания программ, которые исполняются вычислительным блоком конвейера. В рассматриваемом языке присутствуют следующие классы инструкций: инструкции для работы с регистром, инструкции арифметических операций, инструкции битовых операций, инструкции для условного и безусловного перехода на метку, инструкция записи в регистр выходного порта.

Важной особенностью также является отсутствие отдельной области памяти для данных программы. Данные задаются в виде аргументов команд и являются неотделимой частью машинных команд.

2. Постановка задачи

Рассматривается коммутатор, построенный на базе сетевого процессора (RuNPU), архитектура которого описана в разделе 1.1, для управления коммутатором используется протокол OpenFlow. Сетевой процессор работает под управлением программы на языке ассемблера, которая получается путём трансляции таблиц потоков OpenFlow. Для трансляции ис пользуется система трансляции таблиц потоков.

Пусть имеется таблица потоков OpenFlow, содержащая в себе правила с признаками с точным совпадением и маскирующие признаки. Данная таблица транслируется в программу обработки сетевых пакетов на языке ассемблера. СПУ выполняет классификацию полученного пакета по полученной программе на языке ассемблера. Необходимо разработать алгоритм сжатия таблиц потоков, который должен удовлетворять следующим условиям:

- программа, полученная путём трансляции сжатой таблицы потоков, должна занимать меньше памяти, чем программа, полученная из исходной таблицы потоков;
- программа, получаемая из сжатой таблицы потоков, должна быть идентична программе, полученной из исходной таблицы потоков.

3. Имитационная модель сетевого процессора

Имитационная модель СПУ программа, написанная на языке программирования *python*3, которая позволяет проводить имитационное моделирование работы СПУ. Имитационная модель может получать пакеты на любой из виртуальных 24-х портов, обрабатывать пакет, и отправлять пакет на выходные порты. Имитационная модель СПУ позволяет оценить количество тактов, затраченных на обработку пакета, а также объём памяти, затраченный на программу обработки сетевых пакетов. На вход имитационной модели СПУ подаётся программа обработки сетевых пакетов на языке ассемблера, набор входных сетевых пакетов. На выходе имитационной модели получаем записи обработанных пакетов и статистическую информацию.

Работа имитационной модели СПУ состоит из следующих шагов:

1. **Макрогенерация.** Производится замена констант, объявленных с помощью директив define, на их числовые значения. Происходит генерация деревьев поиска из значений, записанных в файлах деревьев поиска. При этом директивы включения деревьев поиска заменяются на соответствующие наборы команд и меток перехода.
2. **Запись программы во внутреннее представление.** Команды полученной на предыдущем шаге программы переводятся во внутреннее представление программы

имитационной модели, моделирующее адресное пространство вычислительного конвейера.

3. **Обработка входных пакетов.** Происходит последовательная обработка входных пакетов с записью выходных пакетов. Также на этом шаге в режиме отладки возможна запись информации о последовательности выполнения команд.

4. **Вывод статистической информации.** По результатам обработки входных пакетов выводится количество обработанных пакетов, среднее число тактов, затраченное на обработку одного пакета, объем занимаемой программой памяти, данные по энергопотреблению и др.

Имитационная модель СПУ состоит из следующих основных модулей (рис. 1a): **Application** – главный модуль имитационной модели, который служит для управления другими модулями; **Pipeline** – модуль, отвечающий за эмуляцию работы конвейера СПУ; **In/OutFIFO** – модули отвечающие за эмуляцию работы входных и выходных очередей СПУ; **Decision Engine (DE)** – модуль, отвечающий за эмуляцию работы одного вычислительного блока СПУ.

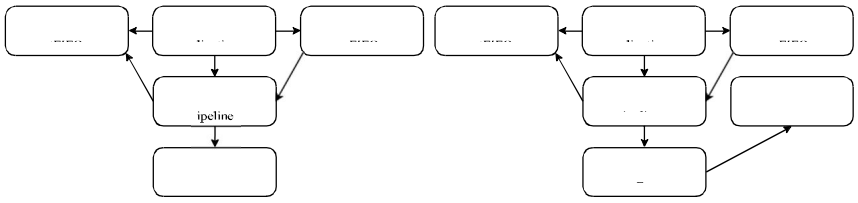

Рис. 1. Архитектура имитационной модели
(a) Исходная архитектура имитационной модели СПУ (b) Изменённая архитектура имитационной модели СПУ

Для работы с алгоритмами сжатия в имитационную модель СПУ необходимо было внести изменения. А именно, в модуль **DE** добавить инструкцию cpucall обращения к ЦПУ коммутатора для получения информации о части таблицы потоков, которая не хранится в СПУ на данный момент. Также необходимо было добавить модуль **CPU** (рис. 1b), который отвечает за эмуляцию ответа ЦПУ при вызове инструкции cpucall.

4. Система трансляции таблиц потоков в язык ассемблера сетевого процессорного устройства

Рассмотрим структуры данных, применяемые в трансляторе таблиц потоков OpenFlow. Они необходимы для корректного промежуточного представления таблиц потоков при процессе трансляции.

4.1 Структура данных для представления таблиц потоков в виде дерева

В системе трансляции для представления таблицы потоков с набором правил R используется дерево с помеченными вершинами и дугами. С каждой вершиной дерева, кроме вершин листьев, связаны следующие значения: признак из множества рассматриваемых признаков; подмножество набора правил R.

Структура данных строится по следующим правилам, где v – вершина дерева, которой соответствует признак m и подмножество правил S R: корню дерева соответствует всё множество правил R; если M множество всевозможных значений признака m в правилах из подмножества S, то для каждого значения f M у вершины v существует потомок, к которому ведёт дуга с пометкой f ; если вершины u- потомок вершины v в которую ведёт дуга с пометкой f , то подмножество правил вершины u состоит только из правил в S, у которых значение признака m равно f . Описанная структура данных позволяет выполнять поиск идентифицирующего обрабатываемый пакет правила в таблице потоков.

4.2 Структура данных для представления таблиц потоков в виде АВЛ дерева

В предыдущей работе [6] была разработана структура данных для представления таблиц классификации в виде АВЛ дерева. Аналогично таблицу потоков можно также представить в виде АВЛ дерева, с каждой вершиной АВЛ дерева связаны следующие

значения: скалярное значение соответствующее набору признаков $I = m_1, m_2, \ldots, m_k$, вычисляемое по расширенному алгоритму представления префиксов как скалярных величин; подмножество набора правил R, соответствующих данному набору признаков.
Таблица 1. Сравнение алгоритмов сжатия

Название	**Сложность построения**	**Степень сжатия**	**Внешняя память**	**Необходимость декомпрессии**
Распространённые алгоритмы	$O(K * \log_2 N)$	0.1 . . . 0.8	нет	да
Оптимальное кеширование	$O(N^2)$	0.1 . . . 0.9	да	нет
Рекурсивное сокращение	$O(N * log(N))$	0.1	нет	нет
Использование битовых строк	$O(\frac{W}{K} * L)$	0.5	нет	нет

4.3 Структура данных для промежуточного представления таблицы потоков

Для применения алгоритмов сжатия необходимо промежуточное представление таблицы потоков. А именно в дополнение к существующим структурам данных к каждой вершине необходимо добавить следующие значения: величину, соответствующую сумме частот использования каждого правила из подмножества правил данной вершины, величина, соответствующая сумме частот использования потомков данной вершины, если у данной вершины нет потомков, то данные величины равны.

4.4 Алгоритмы сжатия

У каждого рассмотренного алгоритма сжатия есть свои достоинства и недостатки (таблица 1). Исходя из качеств каждого алгоритма сжатия для дальнейшей реализации были выбраны два алгоритма сжатия данных. Алгоритм оптимального кеширования, так как он имеет наилучшую возможную степень сжатия из всех алгоритмов, и алгоритм рекурсивного сокращения, который имеет почти такую же степень сжатия, что и алгоритм оптимального кеширования, но при этом не требует выполнения обращений к ЦПУ.

4.4.1 Алгоритм предварительной оптимизации таблицы потоков

Данный алгоритм служит для предварительной оптимизации таблиц потоков перед их сжатием выбранными алгоритмами сжатия. Основная идея данного алгоритма заключается в удалении незначащих и дублирующихся правил из таблицы потоков. На вход алгоритму предварительной оптимизации подаётся корень дерева, построенного по таблице потоков. Основным принципом является обход в глубину, по средствам которого выбираются похожие правила в дереве, затем все листовые вершины, в которых не осталось правил, удаляются.

4.4.2 Алгоритм оптимального кеширования

Основной частью алгоритма оптимального кеширования [5] является операция разделения исходной таблицы потоков на две. Для применения дальнейшего алгоритма необходимо построить изначальное дерево с дополнительными полями в узлах. Получить набор вершин дерева, отсортировать этот набор в невозрастающем порядке сумм вероятностей вершин. Создать второй набор вершин, из которого в последствии будет строится дерево. Создать счётчик, хранящий сумму вероятностей вершин во втором наборе вершин. Получить первую вершину с максимальной суммой вероятностей, увеличить счётчик на его вероятность, добавив эту вершину во второй набор вершин и удалив из первого. Повторять последние три операции, пока счётчик меньше 0.95. Построить дерево из второго набора вершин.

После выполнения этих операций мы получаем два набора вершин, первый из которых отвечает за второстепенное дерево, а второй за первостепенное.

Первостепенное дерево преобразуется в программу на языке ассемблера, которая загружается в СПУ. А второстепенное дерево хранится на ЦПУ, доступ к нему осуществляется по средствам операции cpucall.

4.4.3 Алгоритм рекурсивного сокращения

Данный алгоритм основан на применении дерева HyperSplit [7], а сжатие производится за счёт удаления дублирующихся правил [8]. Под дублирующимися правилами в таблице потоков понимаются следующие правила: правило, содержащееся в вершине дублируется правилом в вершине, являющейся листом для данной вершины (частично дублирующиеся правило); правило, содержащееся в вершине дублируется правилами во всех вершинах, являющихся листьями для данной вершины (полностью дублирующиеся правило). Соответственно дублирующиеся правила перемещаются наверх дерева, что позволяет удалить полностью дублирующиеся правила.

Данный алгоритм рекурсивно использует NewHypersplit [9] [10] для удаления повторяющихся правил из дерева.

5. Экспериментальное исследование реализованных алго ритмов сжатия

При проведении экспериментального исследования ставилась цель получить оценку степени сжатия программы на языке ассемблера сетевого процессорного устройства, на различных данных и оценку времени обновления таблиц потоков при использовании алгоритмов сжатия, на различных данных.

5.1 Методика экспериментального исследования

Для оценки параметров необходимо исследовать программу на языке ассемблера, получаемую при использовании системы трансляции с алгоритмами сжатия и без. Для каждой программы, с помощью имитационной модели СПУ, будут исследоваться такие параметры как: объём памяти занимаемой программой при обработке пакетов на имитационной модели сетевого процессорного устройства; среднее время обработки пакета в тактах СПУ.

Для проведения экспериментального исследования, необходимо последовательно выполнять следующие действия для каждого набора входных данных:

1. выбрать таблицу потоков для данного эксперимента;
2. провести трансляцию выбранной таблицы потоков в программу на языке ассемблера:
   - без использования алгоритмов сжатия, обычное дерево;
   - без использования алгоритмов сжатия, с АВЛ деревом;
   - с использованием разработанных алгоритмов сжатия;
3. провести эмуляцию работы сетевого процессорного устройства с полученными программами на языке ассемблера;
4. провести оценку результатов, полученных в данном эксперименте.

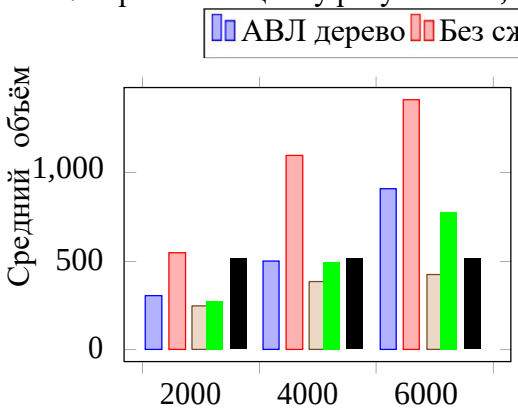

Рис. 2а. Зависимость объёма используемой памяти СПУ от количества правил

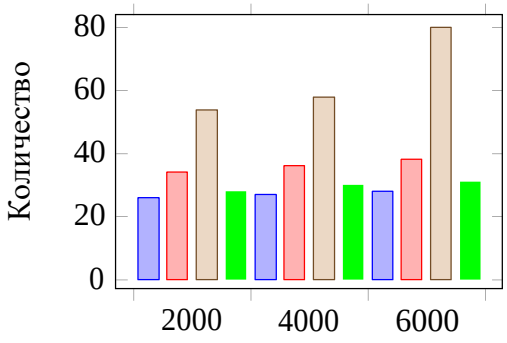

Рис. 2б. Зависимость среднего количества инструкций на обработку одного пакета от количества правил

5.2 Результаты экспериментального исследования

Было проведено экспериментальное исследование реализованных алгоритмов сжатия на имитационной модели СПУ. В ходе экспериментального исследования было выявлено, что использование реализованных алгоритмов сжатия позволяет сократить использование памяти СПУ.

В экспериментальном исследовании были использованы таблицы потоков различных размеров. Максимальный размер таблицы потоков составлял 6000 правил. По результатам экспериментального исследования видно, что без сжатия на СПУ могут хранится таблицы потоков размером до 1500 правил.

Алгоритм оптимального кеширования показал наилучший коэффициент сжатия (рис. 2a), но при этом на обработку данного пакета в среднем затрачивалось наибольшее количество инструкций (рис. 2b). Это вызвано необходимостью обращения СПУ к ЦПУ, для получения правил, которые не хранятся в памяти СПУ.

Алгоритм рекурсивного сокращения показал коэффициент сжатия хуже, чему алгоритм оптимального кеширования (рис. 2a), но меньшее среднее количество инструкций на обработку одного пакета (рис. 2b).

Заключение

По результатам работы можно говорить об уменьшении объёма используемой памяти СПУ, для хранения программ на языке ассемблера, что позволило увеличить возможное количество правил в таблицах потоках. Благодаря алгоритмам сжатия, которые были реализованы в системе трансляции таблиц потоков, наилучший результат показал алгоритм оптимального кеширования. Также были внесены изменения в существующую имитационную модель СПУ, что позволило использовать его для проведения экспериментального исследования.

В качестве возможных направлений дальнейших исследований можно указать изучение возможности применения более продвинутых алгоритмов сжатия с использованием специализированной TCAM памяти.

Работа выполнена при частичной поддержке РФФИ, грант № 19-07-01076.

Литература

1. **Р.Л. Смелянский.** Программно-конфигурируемые сети. В: Открытые системы 9 (2012). С. 15-26.
2. **Беззубцев С.О., Васин В.В., Волканов Д.Ю., Жайлауова Ш.Р., Мирошник В.А., Скобцова Ю.А., Смелянский Р.Л.** Об одном подходе к построению сетевого процессорного устройства //Моделирование и анализ информационных систем. – 2019. Т. 26, № 1. С. 39-62.
3. Open Networking Foundation. OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04). 2012.
4. **Gábor Rétvári Tapolcai J., Kőrösi A., Majdán A., Heszberger Z.** Compressing IP forwarding tables: Towards entropy bounds and beyond. В: ACM SIGCOMM Computer Communication Review 43.4 (2013). С. 111-122.
5. **Ori Rottenstreich и János Tapolcai.** Optimal rule caching and lossy compression for longest prefix matching. В: IEEE/ACM Transactions on Networking 25.2 (2016), с. 864-878.
6. **Н.И. Никифоров, Д.Ю. Волканов, Ю.А. Скобцова.** Анализ и исследование структур данных для поиска в таблицах классификации в сетевом процессорном устройстве с архитектурой RuNPU // Программные системы и инструменты. Тематический сборник № 20. Москва, 2020. С. 118-132.
7. **Yaxuan Qi и др.** Packet Classification Algorithms: From Theory to Practice. В: май 2009. С. 648-656.

8. **Yeim-Kuan Chang и Han-Chen Chen.** Fast packet classification using recursive endpoint- cutting and bucket compression on FPGA. B: The Computer Journal 62.2 (2019), с. 198-214.
9. **Pankaj Gupta и Nick McKeown.** Packet classification using hierarchical intelligent cuttings. B: Hot Interconnects VII. T. 40. 1999.
10. **Sumeet Singh и др.** Packet classification using multidimensional cutting. B: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. 2003, с. 213-224.