ПРИМЕНЕНИЕ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ПРИ РАЗРАБОТКЕ АРХИТЕКТУРЫ СЕТЕВОГО ПРОЦЕССОРНОГО УСТРОЙСТВА С ХРАНЕНИЕМ СОСТОЯНИЯ

Я.К. Кузьмин, Д.Ю. Волканов (Москва)

1. Введение

В настоящее время активно развиваются программно конфигурируемые сети (ПКС) [1]. Основой технологии ПКС является перенос управляющих функций сети из сетевых устройств на отдельное устройство, называемое контроллером. При таком подходе контроллер выполняет централизованное управление сетью путём отправки команд коммутаторам. В свою очередь, коммутаторы выполняют обработку пакетов на основе правил, полученных от контроллера. Таким образом, во всех случаях, требующих изменения набора правил обработки пакетов (например, появление нового потока пакетов), сетевое устройство обращается к контроллеру.

Сетевое процессорное устройство (СПУ) – это специализированная интегральная схема, предназначенная для обработки пакетов в сетевых устройствах, например, в коммутаторах. СПУ выполняет следующие функции [2]:

- получение пакета с физического уровня;
- выделение заголовка пакета для дальнейшей классификации;
- классификация пакета (идентификация пакета по заголовку с целью определения набора правил обработки, которые должны быть применены к пакету);
- модификация заголовка пакета и принятие решения о пути следования пакета;
- управление очередями;
- передача пакета на физический уровень.

Программируемые СПУ – это тип СПУ, позволяющих загружать новые алгоритмы обработки пакетов и определять новые сетевые протоколы в процессе работы при помощи обновления записанной в СПУ программы [3].

При проектировании СПУ для исследования характеристик разработанной архитектуры применяется имитационное моделирование. Для разработанной архитектуры создаётся имитационная модель, позволяющая проанализировать особенности работы СПУ, оценить пропускную способность, энергопотребление, требуемую площадь чипа.

В настоящее время активно развиваются многопоточные транспортные протоколы, такие как MPTCP [4]. Использование многопоточных транспортных протоколов позволяет достичь большей скорости передачи данных благодаря более полному использованию ресурсов сети. Для наиболее полного использования ресурсов сети необходимо использовать разные маршруты для разных транспортных потоков и балансировать потоки между маршрутами. Но в ПКС алгоритмы балансировки транспортных потоков создают высокую нагрузку на контроллер, так как являются алгоритмами обработки пакетов, требующими хранения состояния [5]. Решением данной проблемы является перенос состояния алгоритма балансировки транспортных потоков с контроллера на коммутатор.

В данной работе рассматриваются модификации существующей архитектуры СПУ RuNPU, позволяющие использовать алгоритмы обработки пакетов с хранением состояния, требующие синхронизации состояния между портами СПУ.

Рассматривается имитационная модель СПУ RuNPU [6] и её применение для исследования характеристик СПУ в ходе работы по внесению модификаций в его архитектуру для поддержки новых алгоритмов. Описана структура имитационной модели, путь, по которому проходит сетевой пакет при обработке, описаны изменения, внесённые в имитационную модель.

Статья состоит из следующих разделов: постановка задачи, описание имитационной модели СПУ, описание модификаций имитационной модели СПУ, описание экспериментального исследования.

2. Постановка задачи

Дана архитектура СПУ RuNPU. Архитектура основана на системе вычислительных конвейеров. На каждой стадии конвейера расположено устройство памяти и процессорное ядро. Архитектура СПУ имеет следующие особенности:

- каждый порт СПУ имеет отдельный вычислительный конвейер;
- конвейеры не связаны друг с другом и работают параллельно, синхронно;
- конвейеры имеют одинаковое число стадий;
- время обработки пакета стадией конвейера не превышает 250 тактов.

Состояние алгоритма обработки пакетов представляет собой набор переменных, доступных на чтение и запись для программы, загруженной в СПУ, значение которых сохраняется при переходе к обработке следующего пакета.

Требуется предложить метод, обеспечивающий синхронизацию состояния алгоритма обработки пакетов между конвейерами СПУ с учётом особенностей архитектуры СПУ.

3. Имитационная модель СПУ RuNPU

Для оценки характеристик архитектуры СПУ проведено экспериментальное исследование. Для проведения экспериментального исследования было разработано программное средство, представляющее собой имитационную модель СПУ [6] (Рис. 1). За основу было взято программное средство, представляющее собой имитационную модель исходной архитектуры СПУ RuNPU. Данная имитационная модель написана на языке программирования Python.

В ходе работы имитационная модель позволяет получить следующую информацию: среднее количество тактов на обработку пакета, пропускную способность, среднее энергопотребление. Имитационная модель состоит из следующих модулей: **Application** – основной модуль программы. **Pipeline** – модуль, имитирующий работу конвейера. **InFIFO** – модуль, имитирующий работу входной очереди. В данном модуле происходит чтение pcap файлов с пакетами, подаваемыми на входные порты. Формат pcap позволяет хранить последовательность сетевых пакетов в файле. **OutFIFO** – модуль, имитирующий работу выходной очереди. В данном модуле происходит запись pcap файла, соответствующего данному порту. **DE** (Decision Engine). Данный модуль имитирует работу вычислительного ядра стадии конвейера. **Memory** – модуль, имитирующий работу устройства памяти, находящегося на стадии конвейера. **Context** – модуль, хранящий состояние вычислительного ядра (адрес текущей инструкции,

значение аккумулятора, регистра сдвига). **Packet memory** – модуль, имитирующий работу устройства памяти, используемого для хранения тел пакетов.

Рис. 1: Схема работы имитационной модели

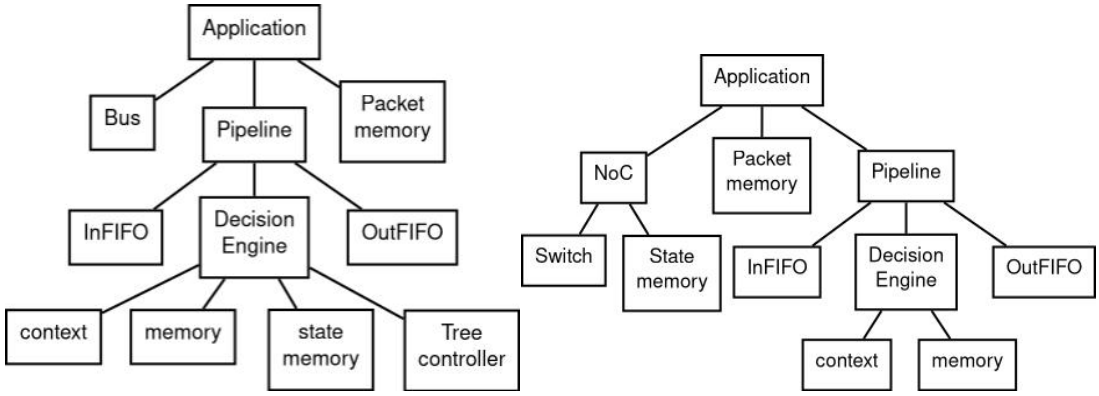

Рис. 2: а) Схема связи модулей имитационной модели с общей шиной. б) Схема связи модулей имитационной модели с сетью с объединением запросов.

Имитационная модель имитирует потактовую работу модулей. Каждый такт происходит вызов метода tick каждого модуля имитационной модели. При этом каждый модуль выполняет действия, соответствующие данному такту: Модули DE выполняют следующую инструкцию программы обработки пакетов, InFIFO получает следующий пакет из входной очереди и передаёт его на вход вычислительного конвейера, OutFIFO получает обработанный пакет и записывает его в соответствующий выходной порт. Таким образом, пакет проходит следующий путь: пакет формируется модулем InFIFO и передаётся на вход соответствующего вычислительного конвейера.

На каждой стадии конвейера происходит выполнение программы, записанной в модуле Memory данной стадии конвейера. В соответствии с программой происходит модификация заголовка пакета и его метаданных. После обработки на вычислительном конвейере пакет передаётся в модули OutFIFO, соответствующие выходным портам, через которые пакет должен быть отправлен. Маска выходных портов содержится в метаданных пакета. Модули OutFIFO записывают полученные пакеты в соответствующие pcap файлы.

4. Предлагаемые модификации архитектуры СПУ и имитационной модели

4.1 Общая шина

В данном методе предлагаются следующие модификации архитектуры СПУ: стадии конвейеров одинаковой глубины связываются общей шиной. Шина используется для синхронизации данных в устройствах памяти для хранения состояния на стадиях конвейера. При записи в устройство памяти по шине передаётся сообщение, содержащее адрес и новое значение соответствующей ячейки памяти. Процессорному ядру доступны две операции с памятью для хранения состояния: чтение из памяти по произвольному адресу и запись в память по произвольному адресу.

При чтении из памяти происходит извлечение значение из соответствующей ячейки локального для данного ядра устройства памяти. При записи значения в память помимо записи в локальное устройство памяти новое значение передаётся остальным устройствам памяти по шине. Таким образом, при получении сообщения со стороны шины устройство памяти должно обновить значение соответствующей ячейки памяти.

К шине, соединяющей устройства памяти, предъявляется следующее требование: данные должны быть переданы за один такт, иначе возникает возможность одновременной модификации ячеек памяти с одинаковым адресом без возможности повтора инструкции. На рисунке показан процесс работы при одновременной записи в ячейку памяти в одном ядре и чтении значения этой ячейки в другом ядре.

Однако не во всех случаях повтор инструкции является приемлемым решением. Для более полного контроля над коллизиями в памяти вводятся две инструкции для работы с устройством памяти для хранения состояния:

• **stread** – чтение из памяти по адресу, заданному в аккумуляторе. При блокировке операции из-за обновления этой ячейки памяти происходит переход по адресу, указанному в параметре инструкции;

• **stwrite** – запись в память по адресу, заданному в первом параметре инструкции. При блокировке операции из-за обновления этой ячейки памяти происходит переход по адресу, указанному во втором параметре инструкции.

В случае работы с памятью с использованием блока tree controller при обнаружении изменения текущей ячейки памяти в процессе поиска блок заново считывает данные из памяти.

Для реализации данного подхода в имитационную модель были добавлены следующие модули (Рис. 2 а):

• **State memory** – модуль, реализующий функциональность устройств памяти для хранения состояния;

• **Bus** — модуль, моделирующий работу шины, связывающий устройства памяти разных конвейеров;

• **Tree controller** – модуль, моделирующий работу блока tree controller, управляющего деревом поиска, находящегося в памяти для хранения состояния.

4.2 Сеть передачи данных

Данный подход предполагает соединение устройств памяти для хранения состояния и процессорных ядер при помощи сети коммутации пакетов со специализированными коммутаторами, позволяющими выполнять операцию объединения запросов. В качестве сети, соединяющей процессорные ядра и устройства памяти, рассматривается Ω-сеть [7]. Сеть разделена на слои, коммутатор на каждом уровне сети имеет четыре соединения: по два для предыдущего слоя и следующего слоя. Каналы связи между слоями строятся по следующему правилу: пусть $s_1 s_2 \ldots s_{n-1} s_n$ – это двоичная запись номера некоторого выхода слоя k. Тогда этот выход соединён с таким входом слоя k+1, номер которого имеет двоичное представление $s_2 \ldots s_{n-1} s_n s_1$.

Количество слоёв (и количество коммутаторов, через которые необходимо пройти пакету) в Ω-сети составляет $\log_2(N)$, где N – количество входов сети.

Листинг 1: Модифицированная операция чтение-изменение-запись (добавлен параметр для функции преобразования)

```
function RMW( X,  f,  parameter)
{
temp = X;
X = f( X,  parameter);
return  temp ;
}
```

Запрос к памяти имеет вид чтение-изменение-запись. Таким образом, при обращении к памяти передаётся тройка (*id, addr, f*), где *id* – индивидуальный номер запроса (должен быть уникальным), *addr* — адрес ячейки памяти, к которой обращён запрос, *f* – идентификтор функции, применяемой к ячейке памяти. В качестве возвращаемого значения операции доступа к памяти используется значение, находившееся в ячейке памяти до выполнения операции преобразования, соответствующей запросу.

Данный подход требует размещения у каждого устройства памяти и коммутаторов арифметико-логических устройств, способных выполнять функции преобразования.

Для работы с памятью необходим набор операций. В данной архитектуре есть два подхода к работе с памятью для хранения состояния. Первый подход заключается в прямой работе с ячейками памяти для хранения состояния, обращение к ним происходит по адресу. Второй подход предполагает хранение двоичного дерева поиска в памяти для хранения состояния и работа с ним при помощи специального блока управления деревом поиска tree controller, соединённого с устройством памяти. Набор операций должен учитывать эти подходы.

В виде, в котором данный подход рассмотрен в статье, отсутствует возможность выполнять операции, которые требуют передачи параметра функции преобразования, например, присваивание. В качестве решения можно предложить добавление к запросу к памяти нового поля, содержащего параметр функции преобразования: (*id, addr, f, parameter*). В этом случае выполнение функции преобразования примет вид, представленный на листинге 1.

Для реализации данного подхода в имитационную модель были добавлены следующие модули (рис. 2 б):

- **State memory** – модуль, реализующий функциональность устройств памяти для хранения состояния;
- **NoC** – модуль, имитирующий работу сети передачи данных, связывающей устройства памяти и вычислительные ядра на стадиях конвейеров;
- **Switch** – модуль, имитирующий работу коммутатора в сети реализуемой модулем NoC.

5. Экспериментальное исследование

5.1 Цель экспериментального исследования

Цель экспериментального исследования — проверить, приводят ли предложенные модификации к нарушению ограничений архитектуры СПУ, и оценить

характеристики модифицированной архитектуры СПУ, такие как скорость обработки пакетов.

5.2 Методика экспериментального исследования

Для проведения экспериментального исследования разработанной архитектуры используется имитационная модель СПУ, написанная на языке программирования Python, описанная в разделе 3. В ходе модификации имитационной модели в неё были добавлены модули, моделирующие работу системы синхронизации состояния алгоритма обработки пакетов. Для каждого из двух подходов реализован свой набор модификаций. Для подхода с общей шиной добавлены модули, моделирующие работу шины и устройств памяти для хранения состояния алгоритма обработки пакетов. Для combining mechanism добавлены модули, моделирующие работу сети, коммутаторов и устройств памяти.

Для исследования написана программа на языке ассемблера для СПУ, реализующая алгоритм flowlet switching [8], используемый для балансировки транспортных потоков. Данный алгоритм был выбран, так как он является алгоритмом обработки пакетов с хранением состояния и для него требуется обеспечить синхронизацию состояния между портами СПУ.

В ходе исследования на вход имитационной модели подаются текстовый файл с текстом программы на языке ассемблера и pcap файлы с сетевыми пакетами, подаваемыми на входные порты СПУ, каждый файл соответствует своему входному порту.

Для проведения исследования были сгенерированы пакеты, представляющие собой набор транспортных потоков, в отдельные pcap файлы записаны пакеты, подаваемые на соответствующие порты СПУ. Каждый поток состоит из 16 пакетов. В последовательности пакетов, подаваемой на вход на каждый порт имитационной модели, содержится 500 потоков.

В ходе экспериментального исследования для каждого рассматриваемого подхода производится серия запусков имитационной модели и измеряются характерстики модифицированной архитектуры. Каждый запуск в серии отличается числом конвейеров, одновременно работающих с устройствами памяти для хранения состояния алгоритма обработки пакетов. Серия состоит из 24 запусков, в которых от 1 до 24 конвейеров одновременно работают с устройствами памяти для хранения состояния. На те конвейеры, которые не работают в данном запуске, тестовые пакеты не подаются.

5.3 Результаты экспериментального исследования

5.3.1 Дублирование с обновлением при записи

По итогам исследования количество тактов, требующихся для обработки заголовка пакета, не превышает 250 (Рис. 3 а). Таким образом, данный подход применим.

5.3.2 Подход на основе сети

В ходе исследования данного подхода было обнаружено, что по мере увеличения числа конвейеров, работающих с состоянием, число тактов, требующихся для обработки пакета, быстро растёт и превышает 250 тактов (рис. 3 б), что делает данный подход в текущем виде неприменимым в данном СПУ.

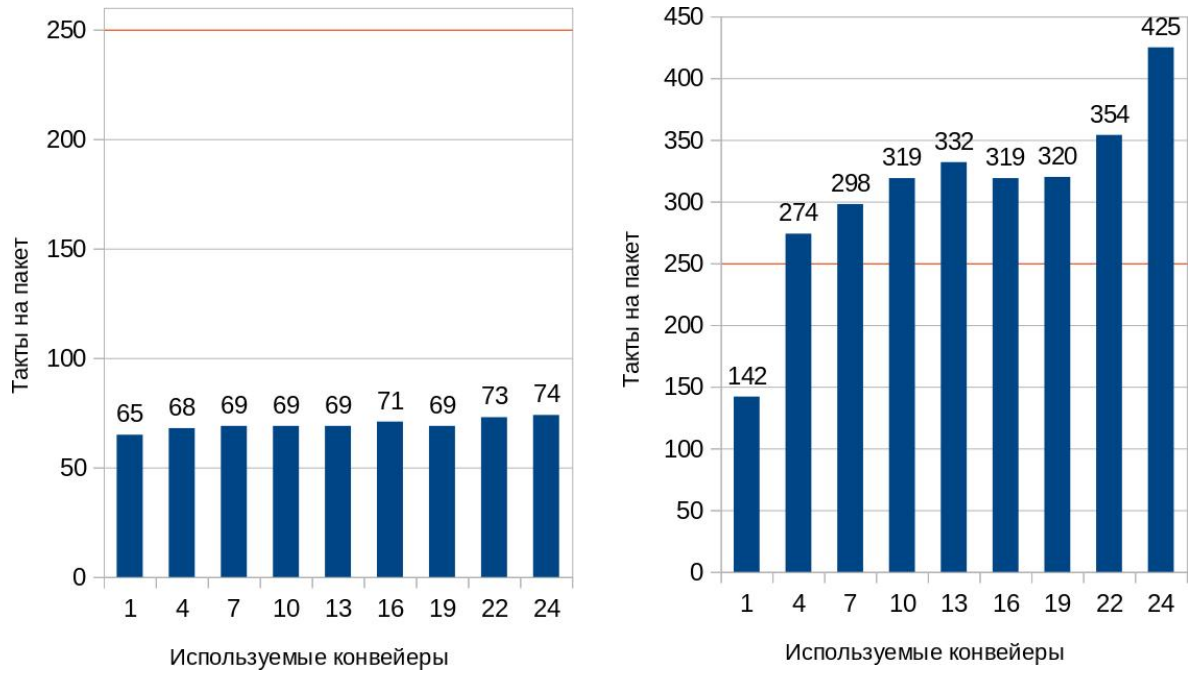

Рис. 3. Результаты экспериментального исследования
а) подхода с общими шинами; б) подхода на основе сети

Заключение

В статье предложены модификации архитектуры СПУ, позволяющие применять алгоритмы обработки пакетов с хранением состояния. Для проведения экспериментального исследования была использована имитационная модель СПУ, потактово моделирующая работу вычислительных конвейеров СПУ. В ходе экспериментального исследования была показана применимость подхода, основанного на общей шине. Подход, основанный на сети передачи данных, не применим в данной архитектуре, так как использование сети приводит к увеличению времени обработки пакета сверх допустимого в рамках рассматриваемой архитектуры.

Дальнейшие исследования могут быть проведены в области оптимизации подхода, основанного на сети, или рассмотрения похожих подходов, так как подход на основе сети позволяет добиться существенной экономии памяти в отличие от подхода, основанного на общих шинах, так как не требует дублирования одних и тех же данных в устройствах памяти на стадиях разных конвейеров.

Литература

1. **Смелянский Р.Л.** Программно-конфигурируемые сети //Открытые системы, № 9, 2012, С. 15-26.
2. **Orphanoudakis T., Perissakis S.** Embedded Multi-Core Processing for Networking // Multi-core Embedded Systems. 2010. CRC Press. P. 399-463.
3. **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.
4. **Ford A., Raiciu C., Handley M., Bonaventure O., Paasch C.** TCP Extensions for MultipathOperation with Multiple Addresses draft-ietf-mptcp-rfc6824bis // IETF RFC-6824. – 2016.
5. **Кузьмин Я.К., Волканов Д.Ю., Скобцова Ю.А.** Исследование применимости алгоритмов обработки пакетов с сохранением состояния в архитектуре сетевого

процессорного устройства RuNPU // Программные Системы и Инструменты – Тематический сборник № 20. Москва, 2020. С. 82-93.

6. **Markoborodov A., Skobtsova Y., Volkanov D.** Representation of the OpenFlow Switch Flow Table //2020 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTec). IEEE, 2020.
7. **Lawrie D.H.** Access and alignment of data in an array processor //IEEE Transactions on Computers. 1975. Т. 100. №. 12. P. 1145-1155.
8. **Cascone C., Pollini L., Sanvito D., Capone A.** Traffic Management Applications for Stateful SDN Data Plane // 2015 Fourth European Workshop on Software Defined Networks. IEEE, 2015. P. 85-90.