Наука и Образование
МГТУ им. Н.Э. Баумана
Сетевое научное издание
ISSN 1994-0408

Наука и Образование. МГТУ им. Н.Э. Баумана.
Электрон. журн. 2015. № 12. С. 165–196.

DOI: **10.7463/1215.0827960**

Представлена в редакцию:	16.11.2015
Исправлена:	30.11.2015

УДК 004.2; 004.31

Аппаратное ускорение выполнения SQL-запросов в MDM-системах на основе МКОД-решения

Подольский В. Э.[1,*], Самочадин А. В.[2], Колосков С. С.[1]

[*] v.e.podolskiy@gmail.com

[1]МГТУ им. Н.Э. Баумана, Москва, Россия
[2]Санкт-Петербургский Политехнический Университет Петра Великого, Санкт-Петербург, Россия

В рамках статьи представлены результаты исследования способов ускорения выполнения SQL-запросов в высоконагруженных системах управления мобильными устройствами (MDM-системах) на примере образовательных организаций высшего образования. В качестве основного решения проблемы большого времени обработки запросов к базам данных в статье представлено использование аппаратного ускорителя. Рассматриваемый в статье аппаратный ускоритель на базе архитектуры ЭВМ со многими потоками команд и одним потоком данных (МКОД), разработанный в МГТУ имени Н.Э. Баумана, обеспечивает выполнение SQL-запросов за счёт реализации аппаратной поддержки операций над структурами данных (множествами). Проведено тестирование образца ЭВМ МКОД при выполнении запросов на наборе макетных данных. В статье приведены результаты определения направлений доработки аппаратного решения ЭВМ МКОД для расширения аппаратной поддержки обработки SQL-запросов, а также описаны способы интеграции решения в MDM-систему вуза. Работы проводятся при финансовой поддержке государства в лице Министерства образования и науки Российской Федерации (государственный контракт № 02.G25.31.0024 от 12.02.2013).

Ключевые слова: управление мобильными устройствами, MDM, МКОД, SQL, обработка структур данных, процессор обработки структур, BYOD

Введение

MDM-системы (Mobile Device Management, MDM – управление мобильными устройствами) служат для организации мобильной инфраструктуры крупных организаций, в том числе университетов. На пользовательском уровне MDM-система представляет собой набор сервисов, предоставляемых конечному пользователю. Тематически сервисы связаны с основными направлениями деятельности организации. Так, в случае образовательной организации, в состав сервисов MDM-системы обычно включают сервис доступа к расписанию, сервис бронирования аудитории, сервис опросов и многое другое. Реализация SOA (service-oriented architecture – сервис-ориентированной архитектуры) в рамках MDM-системы позволяет повысить её привлекательность для конечного пользователя благодаря простоте использования сервисов. Простота использования MDM-систем сейчас обусловлена во многом их тесной интеграцией в мобильные устройства в рамках парадигмы «Bring your own device» (BYOD) [1]. Мобильные устройства, которые весьма широко распространены в повседневной жизни, всё шире применяются и в образовательном процессе не только в вузах, но и в школах [2, 3]. Централизованная MDM-платформа для образовательных целей не является необходимой, но позволяет улучшить ряд параметров организации образовательной деятельности с использованием личных мобильных устройств участников образовательного процесса, в частности будут обеспечены:

- повышение уровня контроля над образовательным процессом;
- упрощение доступа к образовательным материалам;
- единое коммуникационное пространство среди участников образовательного процесса;
- широкий набор предоставляемых в рамках образовательного процесса сервисов.

MDM-платформа является комплексным аппаратно-программным решением. Для развёртывания в образовательной организации сервисов MDM-платформы требуется наличие серверного оборудования, а также сетевой инфраструктуры. Производительность MDM-решений обусловлена качеством используемого аппаратного обеспечения. В связи с тем, что в типовых MDM-системах довольно высока доля операций с базами данных, крайне важной является задача снижения времени выполнения запросов к базам данных программным или аппаратным способами. Наиболее перспективным с точки зрения не только повышения производительности, но и снижения энергозатрат является направление реализации аппаратной поддержки выполнения запросов к базам данных [4, 5, 6, 7].

Повышение производительности выполнения запросов к базам данных при помощи использования аппаратных ускорителей является одним из самых многообещающих на данный момент вариантов, поскольку алгоритмические резервы минимизации времени выполнения запросов к базам данных оказались во многом исчерпаны за последние годы. Использование аппаратного ускорения при этом может происходить по двум направлениям:

- замена программного обеспечения выполнения запросов к базам данных;
- дополнение существующего программного обеспечения.

В реальности любой процесс перехода в многокомпонентных информационных системах на новые аппаратные решения, способствующие улучшению той или иной части процесса обработки данных, требует, как правило, высоких затрат на покупку оборудования и программного обеспечения, его настройку, обучение персонала и так далее. Высокие накладные расходы на интеграцию аппаратных решений (пусть даже и обладающих высокой вычислительной эффективностью), могут остановить процесс перехода на новые аппаратные решения. В этом контексте для компаний важной задачей является обеспечение плавного процесса перехода на новые аппаратные решения, способствующие повышению эффективности решения задач клиента. Таким образом, в ближайшее время наибольшими перспективами обладает направление дополнения существующего программного обеспечения аппаратными решениями, которые могут использоваться для поддержки выполнения отдельных операций над структурами в базах данных.

С точки зрения оптимального сочетания доступности и производительности в рамках задуманного функционала, наиболее подходящими для переходного этапа являются аппаратные решения, реализованные на базе ПЛИС, обладающих интерфейсами для прямого подключения к материнской плате сервера (например, платы с PCIe), а также интерфейсами для подключения к корпоративной сети (оптоволокно, кабель Ethernet). ПЛИС позволяют реализовывать устройства достаточно высокой аппаратной сложности, вплоть до микропроцессоров, исключительно при помощи написания кода на языках VHDL или Verilog. Простота реализации аппаратных решений на ПЛИС, их относительно невысокая стоимость позволяют оперативно апробировать идеи повышения производительности коммерческих приложений за счёт аппаратной поддержки выполнения конкретных операций над данными.

Использование аппаратных ускорителей с принципиально новой архитектурой и набором поддерживаемых команд, оптимизированных для конкретной области применения (например, обработка множеств и других структур данных), позволяет получить ускорение на выполнении запросов к базам данных. К числу подобных аппаратных ускорителей принадлежит вычислительная система со многими потоками команд и одним потоком данных (МКОД), включающая процессор обработки структур, реализующий аппаратное выполнение основных операций над структурами данных из числа операций теоретико-множественного представления. МКОД-система разработана в рамках проекта, реализуемого в МГТУ имени Н.Э. Баумана. Поскольку данная система на аппаратном уровне реализует выполнение команд над множествами, на её основе может быть организована аппаратная поддержка операций и алгоритмов, основанных на обработке множеств [8]. Таким образом, одним из перспективных направлений развития и применения МКОД-системы является поддержка аппаратного ускорения выполнения SQL-запросов к базам данных. В рамках этого направления должен быть разработан базовый набор операций и алгоритмов выполнения SQL-запросов, аппаратно поддерживаемый МКОД-системой и процессором обработки структур.

Принципиальная научная новизна данного исследования заключается в исследовании возможности реализации SQL-запросов к базам данных на базе архитектуры МКОД. Помимо этого, настоящая работа включает в себя **практический аспект** – в ней детально описаны возможности применения ЭВМ МКОД для обработки SQL-запросов в рамках MDM-платформы университета. MDM-платформа университета разработана в рамках государственного контракта № 02.G25.31.0024 от 12.02.2013 и успешно прошла апробацию. Формы запросов, а также структуры данных прошедшей апробации использованы в настоящей работе для проведения тестирования SQL-запросов, адаптированных для МКОД-системы.

1. Платформа управления мобильными устройствами для вуза: архитектура и обработка запросов

Платформа управления мобильными устройствами (MDM-платформа) предоставляет в рамках образовательных организаций высшего образования большое число возможностей по управлению различными сторонами образовательной деятельности студентов, а также по решению задач административного характера. Для реализации этих возможностей в MDM-системе вуза предусмотрены информационные сервисы [9]:

- доступ к расписанию и информации об отмене занятий;
- управление календарём мероприятий;
- доступ к информации об административных процедурах и процессах внутри вуза;
- интерактивные инструменты обучения;
- опросы, тесты и экзамены;
- поддержка групповой работы студентов, распределение заданий;
- поддержка процессов сертификации и защиты выпускной работы;
- мониторинг посещаемости;
- мониторинг успеваемости студентов.

Доступ к информационным сервисам, предоставляемым MDM-платформой, осуществляется при помощи установленного мобильного приложения. Информационные сервисы осуществляют доступ к базам данных, хранящимся на серверах образовательной организации при помощи специализированного web-API. Запросы сервисов диспетчеризуются специальной программой, после чего обрабатываются на сервере – в итоге сервис получает результат обработки запроса и передаёт его пользователю. Обобщённая архитектура MDM-платформы, иллюстрирующая описанный процесс взаимодействия пользователя с сервисами MDM-платформы, представлена на рисунке 1.

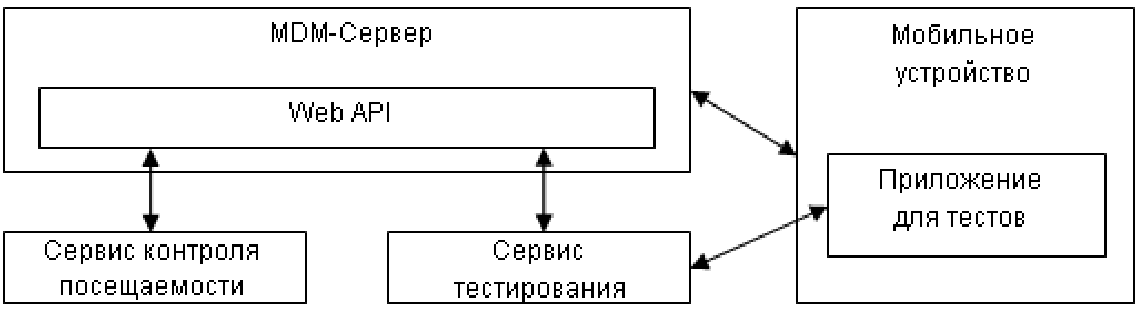

Рис. 1. Обобщённая архитектура MDM-платформы

Платформа управления мобильными устройствами, разрабатываемая в рамках проекта «Создание российского аналога системного программного обеспечения для централизованного управления персональными устройствами и платформами в корпоративных сетях» Санкт-Петербургского политехнического университета и используемая для демонстрации применения МКОД-решения для ускорения обработки данных в рамках MDM-платформы, состоит из следующих взаимосвязанных компонентов:

- **Портал самообслуживания** – решает широкий спектр задач, связанных с диспетчеризацией пользовательских запросов и текстовых сообщений, обработкой запросов различных компонентов платформы, аутентификацией пользовательских устройств. Портал самообслуживания является связующим звеном с остальными компонентами MDM-платформы.
- **Консоль управления** – предназначена для управления учётными записями пользователей мобильных устройств, доступом мобильных устройств к ресурсам, сервисам и приложениям корпоративной сети передачи данных, а также для конфигурирования компонентов программного комплекса.
- **Сервер базы данных** – обеспечивает хранение данных о подключенных к корпоративной сети передачи данных мобильных устройствах пользователей, обеспечивает возможность построения и хранения инвентаризационных отчетов, хранит конфигурационные данные, обеспечивает хранение корпоративных политик для мобильных устройств и установочных файлов корпоративных приложений, а также документов. Помимо этого, выполняет функцию хранения данных журналов работы отдельных компонентов.
- **Сервер мобильного менеджера устройств (ММУ)** – обеспечивает удалённое развёртывание программного обеспечения и групповых политик на мобильных устройствах, обеспечивает работу политик управления мобильными устройствами, обеспечивает актуализацию на мобильных устройства заданных конфигурационных параметров, обеспечивает автоматическое обнаружение подключённых пользовательских устройств, поддерживает возможность блокировки/разблокировки мобильных устройств, поддерживает управление учётными записями пользователей и каталогом корпоративных документов, обеспечивает управление правами на доступ к документам, обеспечивает загрузку документов на мобильные устройства пользователей, обеспечивает доступ мобильных устройств к корпоративным приложениям и другое. В качестве платформы для компонента используется Java EE 8.0, также для работы используется сервер приложений WildFly 8.01.
- **Встраиваемые модули (плагины)** – предназначены для реализации вызовов функций управления и для расширения возможностей менеджера мобильных уст-

ройств, в частности решают задачи обеспечения передачи команд управления мобильными устройствами от Сервера ММУ на мобильные устройства и передачу результатов выполнения команд от мобильных устройств на Сервер ММУ, обеспечение приема от мобильных устройств инвентаризационной информации и информации о текущем статусе и передачу этой информации на Сервер ММУ, обеспечение взаимодействия с мобильными устройствами в процессе передачи на устройства информационных сообщений, документов и других корпоративных данных, предоставление компонентам ММУ информацию о специфических особенностях каждой мобильной платформы, ведение журналов процесса взаимодействия с мобильными устройствами и сохранение журналов на Сервере базы данных ММУ.

- **Сервер балансировки нагрузки** – реализует функционал балансировки нагрузки между развёрнутыми экземплярами компонента «Сервер мобильного менеджера устройств», а также изоляцию сети ММУ от внешних сетей передачи данных и функцию предотвращения DDoS-атак.
- **Сервер рассылки коротких текстовых сообщений** – реализует функцию доставки коротких сообщений, информирования о доставке коротких сообщений, задания параметров доставки сообщений.
- **Программное обеспечение мобильного устройства «Агент ПО ММУ»** – управляет конкретным мобильным устройством, обеспечивая проверку характеристик мобильного устройства минимальным требованиям, проверку наличия новых версий программного обеспечения, проверку возможности использования геолокационных сервисов, получение данных о местоположении устройства, предоставление пользователю возможности просмотра каталога корпоративных документов и самих документов, а также их загрузки и сохранения, предоставление возможности просмотра каталога корпоративных приложений для мобильных устройств, предоставление сервиса просмотра, хранения и удаления коротких текстовых сообщений.

Управление базой данных на сервере БД осуществляется при помощи СУБД PostgreSQL. Для получения данных о пользовательских устройствах, документах, приложениях и иной хранящейся в базе данных MDM-платформы информации используются SQL-запросы. На рисунке 2 представлен пример представления, использующего для построения таблицы SQL-запрос проверки принадлежности пользователя группе или связанным подгруппам. Приведённый запрос реализует алгоритм поиска в глубину на графе связей групп и пользователей. Пример такого графа представлен на рисунке 3.

```
CREATE OR REPLACE VIEW v_user_in_group_membership_include_indirect AS
WITH RECURSIVE search_graph(parent_group_id, user_id, path, cycle) AS
(
      SELECT
            parent_group_id, user_id, ARRAY[(parent_group_id,
            parent_group_id)], false
      FROM user_in_group_membership
      UNION ALL
      SELECT
            e.parent_group_id, sg.user_id, sg.path || (e.parent_group_id,
            e.group_id),
            (e.parent_group_id, e.group_id) = ANY(sg.path)
      FROM search_graph sg INNER JOIN group_in_group_membership e
      ON sg.parent_group_id = e.group_id
      WHERE NOT sg.cycle
)
SELECT DISTINCT parent_group_id, user_id FROM search_graph;

CREATE OR REPLACE RULE
v_user_in_group_membership_include_indirect_delete_rule AS
      ON DELETE TO v_user_in_group_membership_include_indirect DO
INSTEAD();
```

Рис. 2. Пример представления, использующего SQL-запрос поиска в графе

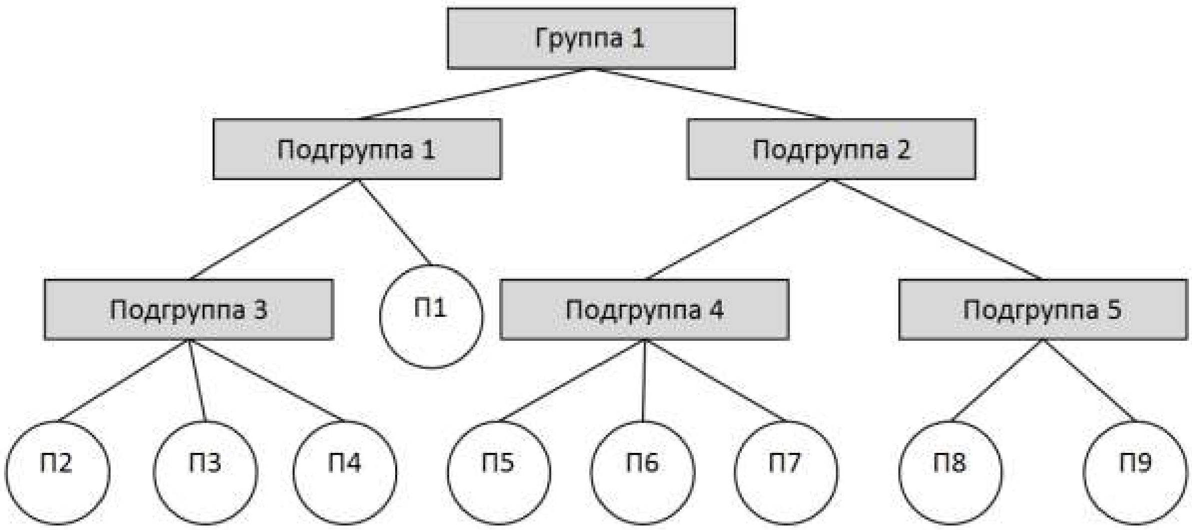

Рис. 3. Граф связей пользователей и групп пользователей (П – узел, обозначающий пользователя)

Обработка данных на сервере базы данных производится с интенсивным использованием алгоритмов обработки структур данных, включая алгоритмы обработки графов. Среди SQL-запросов, обрабатываемых СУБД PostgreSQL в рамках MDM-платформы, особо распространены операторы SELECT, UNION, JOIN (INNER, LEFT OUTER), реализующие операции над множествами элементов данных. Аппаратная поддержка выполнения подобных операций и реализации алгоритмов обработки структур данных способна ускорить процесс обработки СУБД пользовательских запросов.

2. Аппаратная поддержка операций над структурами данных средствами МКОД-системы

Аппаратная поддержка операций над структурами данных реализована в рамках МКОД-системы, разработанной исследовательским коллективом в МГТУ имени Н.Э. Баумана. Данная архитектура реализована в виде двух отдельных процессорных устройств, одно из которых выполняет все команды обработки структур данных, а другое – команды арифметико-логической обработки. Основное отличие предлагаемой архитектуры от существующих аналогов заключается в том, что она реализует обработку единого потока данных при помощи различных потоков команд. Схема взаимодействия устройств в рамках МКОД-системы с топологией общей шины приведена на рисунке 4.

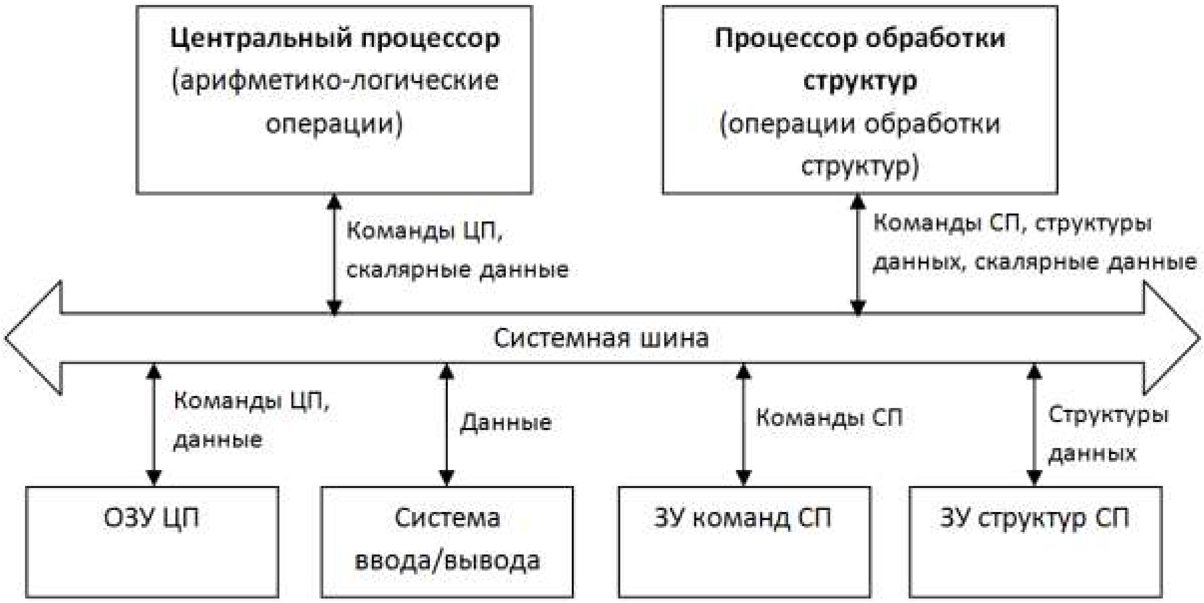

Рис. 4. Схема взаимодействия устройств в рамках МКОД-системы с топологией общей шины

Основными отличительными особенностями МКОД-системы, обеспечивающими аппаратное ускорение выполнения операций обработки структур данных, являются:

- **Параллельное выполнение команд арифметико-логической обработки и команд обработки структур над одним потоком данных.** Параллельное выполнение указанных типов команд - основополагающий принцип архитектуры МКОД. В худшем случае (наличие зависимостей по данным между операциями) возможно последовательное выполнение команды обработки структуры данных и команды арифметико-логической обработки.
- **Аппаратная поддержка выполнения команд над структурами данных процессором обработки структур.** В архитектуре ЭВМ МКОД аппаратно реализовано выполнение команд обработки множеств: поиск, добавление, удаление, удаление структуры, максимум, минимум, мощность, пересечение, объединение, дополнение, срез «меньше, чем», срез «меньше либо равно», срез «больше, чем», срез «больше либо равно», доступ к следующему элементу структуры. Помимо этого,

процессором обработки структур реализуется базовая функция управления потоком команд при помощи операции перехода по тегу.

В основе аппаратной поддержки структур данных лежит B+ дерево. Процессор обработки структур состоит из взаимосвязанных блоков [10]:

- Каталог, хранящий информацию о расположении элементов структуры данных в памяти структур.
- Операционный буфер, служащий для обработки линеек, выбираемых из памяти структур.
- Память структур, являющаяся основным внутренним хранилищем структурной информации и ассоциированных с ней данных.
- Блок загрузки и выгрузки, поддерживающий хранение структур, превышающих по размеру память структур.
- Контролер памяти, осуществляющий управление внешним запоминающим устройством большой ёмкости.
- Буфер данных, получающий информацию от процессора и передающий результаты обработки.
- Устройство управления, обеспечивающие параллельное управление всеми компонентами процессора обработки структур в соответствии с принципами конвейерной обработки.
- Блок выборки команд, обеспечивающий загрузку команд потока обработки структур из локальной памяти, принимающий также синхронизирующие сигналы от ЦП.

Процессор обработки структур реализует набор операций над структурами данных [11]. Реализованные на данный момент операции представлены в таблице 1. R – номер структуры результата, К – ключ, З – значение, С – статус, A,B – номера исходных структур.

Таблица 1. Команды процессора обработки структур

Команда	**Мнемокод**	**Операнды**			**Результаты**		
Поиск	**SEARCH**	R	К	-	С	К	З
Добавление	**INSERT**	R	К	З	С	К	З
Удаление	**DELETE**	R	К	-	С	К	З
Удаление структуры	**DELSTR**	R	-	-	С	-	-
Максимум	**MAX**	R	-	-	С	К	З
Минимум	**MIN**	R	-	-	С	К	З
Мощность	**POWER**	R	-	-	С	К	З
Пересечение	**AND**	R	A	B	С	-	-
Объединение	**OR**	R	A	B	С	-	-
Дополнение	**NOT**	R	A	B	С	-	-
Срез «меньше, чем»	**LS**	R	A	К	С	-	-
Срез «меньше либо равно»	**LSEQ**	R	A	К	С	-	-
Срез «больше, чем»	**GR**	R	A	К	С	-	-
Срез «больше либо равно»	**GREQ**	R	A	К	С	-	-
Следующий	**NEXT**	R	К	-	С	К	З
Переход по тегу	**JT**	Тег	Адрес	-	-	-	-

Для обеспечения максимально возможного ускорения, алгоритм должен быть представлен в том виде, который требуется для его выполнения непосредственно в МКОД-системе. При этом алгоритм может быть представлен как в режиме сопроцессора процессора обработки структур, так и в режиме МКОД. Во втором случае алгоритм, включающий операции обработки структур данных, должен быть разделён на две части: одна часть алгоритма должна содержать только команды обработки структур данных, а другая – только операции арифметико-логической обработки данных. Разделение алгоритма на две такие части может быть произведено при помощи алгоритмов разрезания графа при представлении последовательного алгоритма в виде информационного графа программы с явным указанием на связь команд с используемыми данными для определения зависимостей между командами по данным. Часть алгоритма, отвечающая за обработку структур, должна быть представлена в базисе команд процессора обработки структур.

После разделения алгоритма на две части проводится компиляция каждой из частей, при этом скомпилированный код части программы, отвечающей за арифметико-логическую обработку данных, загружается в ОЗУ центрального процессора, тогда как скомпилированный код части программы, отвечающий за обработку структур данных, загружается в запоминающее устройство команд процессора обработки структур. При этом исходные структуры данных, обрабатываемые в рамках потока обработки структур, загружаются в запоминающее устройство структур процессора обработки структур; результаты обработки структур также сохраняются в данном запоминающем устройстве. Синхронизация обработки данных процессором обработки структур и центральным процессором осуществляется за счёт механизма использования тегов в команде JT (тег – битовое поле, определяющее результат ветвления). Пока значение тега не получено от процессора арифметико-логической обработки (в коде обозначается символом «?»), происходит ожидание передачи тега. Аналогичный подход используется и при ожидании передачи результатов обработки от процессора обработки структур или центрального процессора. Число пересылок данных между потоком обработки структур данных и потоком арифметико-логической обработки определяет скорость выполнения программы в целом.

Реализованный на данный момент в рамках МКОД-системы набор команд процессора обработки структур содержит все необходимые команды для построения любого алгоритма, интенсивно использующего обработку структур. Тем не менее, отдельные специфические операции обработки структур, применяемые в специализированных приложениях (геоинформационные системы, обработка запросов к базам данных и другие), могут быть дополнительно реализованы аппаратно в процессоре обработки структур, помимо базового набора команд.

3. Использование МКОД-системы для ускорения обработки запросов к базам данных MDM-платформы

В рамках реализации функций поддержки мобильной инфраструктуры вуза, MDM-платформа осуществляет интенсивный обмен информацией с внутренними базами данных под управлением СУБД PostgreSQL. Для получения информации из баз данных выполняются сложные SQL-запросы, включающие оператор JOIN, а также рекурсию (при обработке отношений между сущностями базы данных). Операторы SQL и SQL-запросы могут быть представлены в виде отдельных математических операций над множествами или в виде последовательности таких операций. Применение МКОД-системы в аппаратной части MDM-платформы позволит реализовать аппаратную поддержку выполняемых к базе данных запросов на основе аппаратной поддержки операций над множествами. Возможны три вида аппаратной поддержки МКОД выполнения запросов к базам данных на основе SQL:

- **Аппаратная поддержка алгоритмов обработки структур данных, реализуемых в базе данных и включающих несколько SQL-запросов.** Реализация работ по этому направлению может быть выполнена путём полной замены выполняемых алгоритмов обработки структур, хранящихся в базах данных алгоритмами обработки структур, выполняемыми МКОД-системой. Ключевой недостаток данного вида аппаратной поддержки выполнения запросов к базам данных заключается в необходимости ручного написания кода алгоритма, использующего набор запросов, для выполнения в МКОД-системе. Основное преимущество данного варианта состоит в том, что написанный алгоритм будет лучше приспособлен к выполнению на МКОД-системе.
- **Аппаратная поддержка отдельных SQL-запросов.** Данный вид аппаратной поддержки SQL-запросов осуществляется путём замены тела запроса на алгоритм обработки структуры данных, лежащий в основе SQL-запроса, и приспособлением его для обработки в рамках МКОД-системы. Преимущество данного подхода состоит в возможности автоматизации процесса представления SQL-запроса в виде последовательности операторов обработки структур данных и операторов арифметико-логического обработки данных, а также в возможности формирования МКОД-шаблонов типовых SQL-запросов к базам данных MDM-платформы.

- **Аппаратная поддержка отдельных SQL-операторов.** Данный вид аппаратной поддержки основывается на том факте, что большинство SQL-операторов по сути являются реализацией операций над множествами данных. Наглядный пример реализации оператором JOIN операций над множествами представлен на рисунке 5 при помощи диаграмм Виенна.

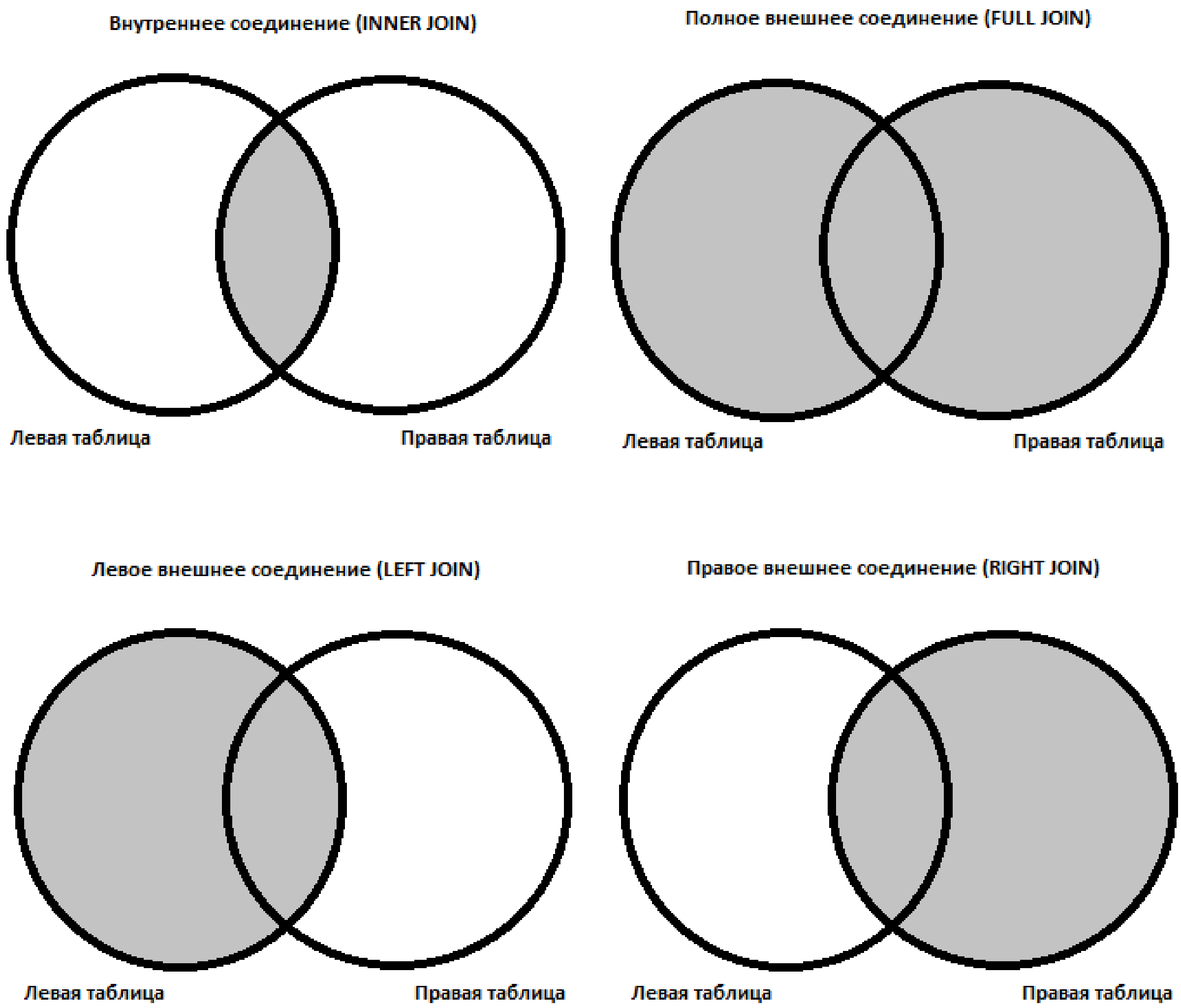

Рис. 5. Наглядный пример реализации оператором JOIN операций над множествами данных

В основе операторов языка SQL лежит реляционная алгебра, основанная на теоретико-множественном подходе. Переход от операторов языка SQL к операциям над множествами представлен в таблице 2.

Таблица 2. Таблица соответствий операторов языка SQL операциям над множествами и командам процессора обработки структур

Оператор языка SQL	Операция над множествами	Команда процессора обработки структур	Пояснение
CREATE	$A \leftarrow A \cup \{x\}$	INSERT	Выполняется последовательность операций вставки. Вставка осуществляется в пустое множество.
DROP	$A \leftarrow \{\emptyset\}$	DELSTR	Удаление таблицы (структуры).
DELETE	$A \leftarrow A \backslash \{x\}$	DELETE	Удаление элемента из таблицы (структуры).
SELECT	-	SEARCH	Выбор элемента.
INSERT	$A \leftarrow A \cup \{x\}$	INSERT	Вставка элемента.
UPDATE	$A \leftarrow A \backslash \{x\}$ $A \leftarrow A \cup \{y\}$	Последовательность DELETE, INSERT	Изменение значения осуществляется удалением старого значения из множества и добавлением нового.
INNER JOIN	$A \cap B$	AND	Внутреннее соединение таблиц.
LEFT JOIN	$C \leftarrow A \cap B$ $C \leftarrow C \cup \big(A \backslash (A \cap B)\big)$	AND, NOT, OR	Использование всех перечисленных операций в соответствии с формулой над множествами: сначала формируется INNER JOIN, после чего добавляются записи из левой таблицы, не вошедшие в INNER JOIN
LEFT OUTER JOIN	$A \backslash (A \cap B)$	AND, NOT	Использование всех перечисленных операций в соответствии с формулой над множествами: сначала формируется INNER JOIN, после чего результат INNER JOIN исключается из «левого» множества A
RIGHT JOIN	$C \leftarrow A \cap B$ $C \leftarrow C \cup \big(B \backslash (A \cap B)\big)$	AND, NOT, OR	Использование всех перечисленных операций в соответствии с формулой над множествами: сначала формируется INNER JOIN, после чего добавляются записи из правой таблицы, не вошедшие в INNER JOIN
RIGHT OUTER JOIN	$B \backslash (A \cap B)$	AND, NOT	Использование всех перечисленных операций в соответствии с формулой над множествами: сначала формируется INNER JOIN, после чего результат INNER JOIN исключается из «правого» множества B
FULL JOIN	$C \leftarrow A \cap B$ $C \leftarrow C \cup \big(A \backslash (A \cap B)\big)$ $C \leftarrow C \cup \big(B \backslash (A \cap B)\big)$	AND, NOT, OR	Использование всех перечисленных операций в соответствии с формулой над множествами: сначала формируется INNER JOIN, после чего добавляются записи из левой и правой таблиц, не вошедшие в INNER JOIN. Обычно не используется.
FULL OUTER JOIN	$A \backslash (A \cap B) \cup B \backslash (A \cap B)$	AND, NOT, OR	Объединение результатов LEFT OUTER JOIN и RIGHT OUTER JOIN
CROSS JOIN	$A\ X\ B$	-	Декартово произведение на данный момент аппаратно не реализовано, но результаты CROSS JOIN могут быть получены при помощи существующего набора команд
UNION	$A \cup B$	OR	Объединение результатов запросов
INTERSECT	$A \cap B$	AND	Пересечение результатов запросов
EXCEPT	$A \backslash B$	NOT	Исключение результатов второго запроса из результатов первого

Аппаратная поддержка реализации отдельного оператора языка SQL или SQL-запроса позволяет заметно ускорить выполнение запросов сервером баз данных. Поскольку разработанное МКОД-решение в настоящий момент реализовано на ПЛИС, физически оно может быть перенесено на сервер при помощи сетевых ПЛИС, например, NetFPGA-1G с Xilinx Virtex-II Pro 50 и 4 интерфейсами 1-гигабитного Ethernet – пример соответствующей архитектуры приведён на рисунке 6. Прошивка ПЛИС МКОД-решением и разработка простейших Ethernet-драйверов позволит в полной мере реализовать аппаратную поддержку обработки SQL-запросов к базам данных. Основным преимуществом плат платформы NetFPGA в рамках разработки способов повышения скорости обработки запросов MDM-системой перед другими сетевыми ПЛИС является ориентация платформы на решение исследовательских задач, в частности поддержка возможности быстрого прототипирования и наличие большого сообщества разработчиков NetFPGA. Тем не менее, существует критическая проблема аппаратной поддержки выполнения SQL-запросов к базам данных с использованием ПЛИС. Эта проблема заключается в малом объёме памяти на борту платы с ПЛИС (к примеру, для NetFPGA 1G объём DDR2 DRAM составляет всего 64 Мб.). В рамках исследовательского проекта обозначенный недостаток не является критическим, но для коммерческого использования при обработке больших массивов данных должно быть предусмотрено решение с большим объёмом памяти.

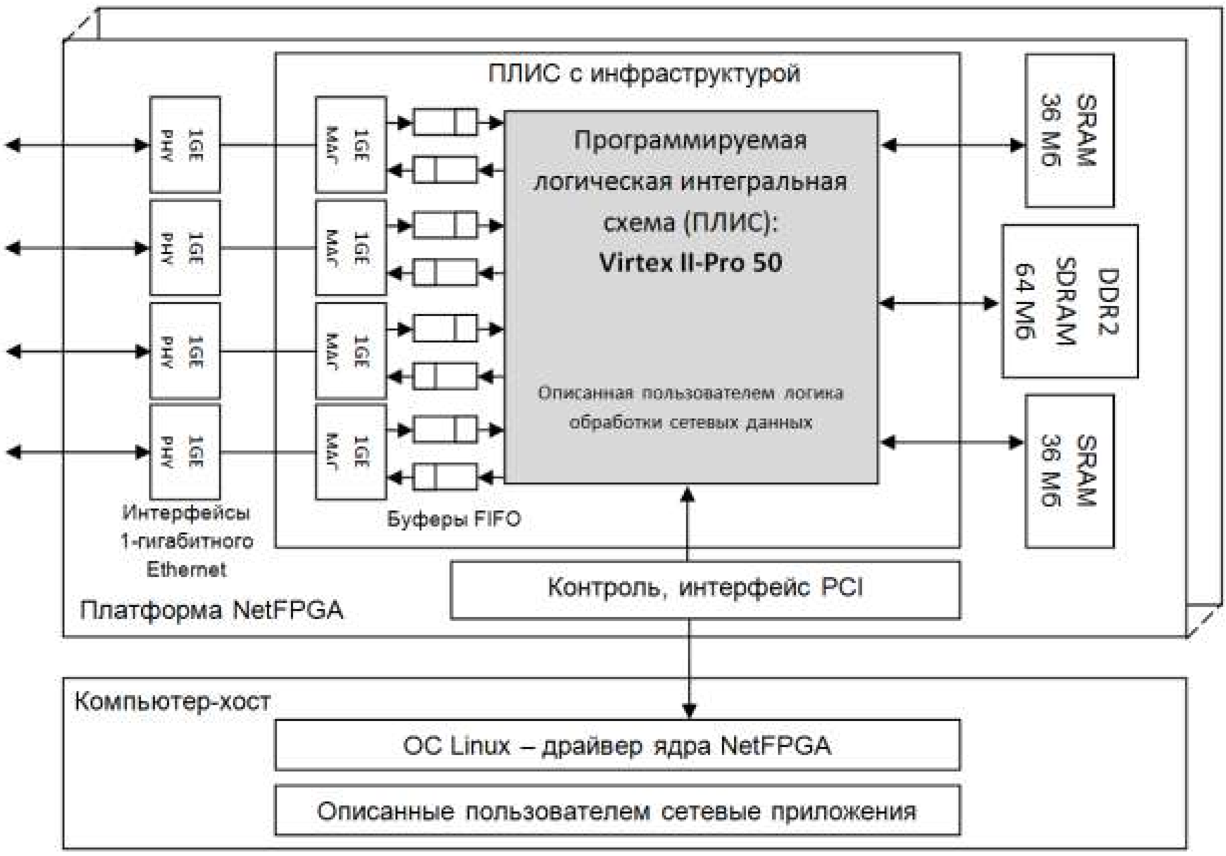

Рис. 6. Архитектурная схема использования платформы NetFPGA в сервере

Плата с ПЛИС с прошитым в ней МКОД-решением помещается в сервер, после чего обработка SQL-запросов к базам данных, хранящимся на сервере, происходит с использованием аппаратно поддерживаемых команд обработки структур данных. Последовательность операций в рамках MDM-платформы с использованием МКОД-решения выглядит следующим образом:

- при помощи мобильного клиента «Агент ПО ММУ» пользователь пытается получить информацию об имеющихся на сервере приложениях;
- действие пользователя преобразуется в SQL-запрос, который диспетчеризуется порталом самообслуживания серверу баз данных;
- SQL-запрос поступает на сервер баз данных, где направляется для обработки на устройство NetFPGA с прошитым МКОД-решением;
- при необходимости в рамках NetFPGA производится предварительная обработка – автоматическое преобразование запроса в формат обработки МКОД-системы;
- в рамках NetFPGA производится обработка запроса прошитым МКОД-решением с использованием баз данных, развёрнутых на сервере;
- результат обработки направляется по сети конечному пользователю (перечень приложений, доступных для установки).

Уменьшение времени выполнения последовательности операций в целом достигается за счёт аппаратного ускорения выполнения SQL-запроса ПЛИС с прошитым МКОД-решением. В следующем разделе впервые приведена реализация базового набора SQL-запросов, покрывающего значительную часть функционала обработки SQL-запросов MDM-платформы.

4. Реализация SQL-запросов в МКОД-системе

Распространённые SQL-запросы могут быть преобразованы в формат обработки МКОД-системой с использованием методики, предложенной в [11]. В данной части статьи приведена реализация базового набора SQL-запросов в формате МКОД. Данная реализация может быть положена в основу создания МКОД-совместимого подмножества языка SQL. Поскольку структурно часть запросов к базам данных MDM-платформы выполнена в виде алгоритмов поиска на графе (в глубину), для ускорения выполнения отдельных таких запросов могут использоваться реализации алгоритмов обхода графа для МКОД [12].

Для реализации различных видов JOIN-запросов могут быть использованы реализуемые процессором обработки структур операции над множествами. В качестве элементов множеств будут выступать ключи в таблицах. Реализацию различных SQL-запросов в МКОД-формате рассмотрим на примере таблиц связей, реализованных в MDM-системе:

- user_in_group_membership – таблица, сопоставляющая идентификатор пользователя (user_id) идентификатору группы, в которой пользователь состоит (parent_group_id);
- group_in_group_membership – таблица, сопоставляющая идентификатор пользовательской группы (group_id) идентификатору другой пользовательской группы (parent_group_id), в состав которой входит пользовательская группа с идентификатором group_id.

Для выполнения операций JOIN на базе МКОД-решения подходит кластерное представление графа, также использованное в [11]. Так, для наших целей таблица user_in_group_membership может быть представлена в виде, показанном в таблице 3. В качестве ключа USR_GRP.KEY для структуры используется идентификатор группы пользователей parent_group_id из исходной таблицы user_in_group_membership. Поле USR_GRP.DATA содержит идентификатор пользователя, входящего в эту группу user_id. В одну группу может входить один и более пользователей.

Таблица 3. Информационная модель для представления таблицы user_in_group_membership

USR_GRP.KEY	**USR_GRP.DATA**
1	3
1,1	54
1,2	33
1,3	12
…	…
n	k
n,1	n_1
n,2	n_2
…	…
n,k	n_k

Таблица group_in_group_membership может быть представлена в виде, показанном в таблице 4. В качестве ключа GRP_PARGRP.KEY для структуры используется идентификатор группы пользователей group_id из исходной таблицы group_in_group_membership. Поле USR_GRP.DATA содержит идентификатор группы parent_group_id , включающей группу из поля group_id. Одна и та же группа с идентификатором parent_group_id может включать в себя несколько групп с идентификаторами group_id.

Таблица 4. Информационная модель для представления таблицы group_in_group_membership

GRP_PARGRP.KEY	GRP_PARGRP.DATA
1	2
1,1	8
1,2	9
…	…
n	k
n,1	8
n,2	10
…	…
n,k	2

Далее приведён псевдокод алгоритмов выполнения разновидностей оператора JOIN. Характерной особенностью большинства стандартных запросов SQL является то, что они используются исключительно для получения некоторых данных, но непосредственно арифметические операции над данными не осуществляют. Эта характерная особенность позволяет реализовать SQL-запрос в МКОД-системе в режиме конвейера, то есть так, чтобы запрос целиком исполнялся лишь с использованием процессора обработки структур.

4.1. Оператор INNER JOIN

Реализация оператора SQL INNER JOIN в рамках МКОД-решения осуществляется с использованием операции AND и SEARCH процессора обработки структур. Алгоритм оператора SQL INNER JOIN для МКОД-системы приведён ниже.

Алгоритм 1 Алгоритм выполнения запроса INNER JOIN / псевдокод алгоритма в конвейерной системе

```
RESULT ← ∅
indexes ← AND(USR_GRP, GRP_PARGRP)        □ Пересечение структур для работы только с группами, которые есть в обеих структурах
key ← NEXT(indexes)
users ← SEARCH(USR_GRP, [key, 0])
ЦИКЛ ПОКА key ≠ ∅
  key ← NEXT(indexes)
  users ← SEARCH(USR_GRP, [key, 0])
  k ← users.DATA
  ЦИКЛ j = 1 to k
    user ← SEARCH(USR_GRP, [key, j])        □ Получаем id пользователя для данной группы пользователей
    parGroups ← SEARCH(GRP_PARGRP, [key, 0])
    m ← parGroups.DATA
    INSERT(RESULT, [user, 0], m)
    ЦИКЛ l = 1 to m
      parGroup ← SEARCH(GRP_PARGRP, [key, l])
      INSERT(RESULT, [user, l], parGroup)        □ Вставка в итоговую структуру. ID пользователя – ключ, значение – ID группы-родителя
    ВСЕ ЦИКЛ
  ВСЕ ЦИКЛ
ВСЕ ЦИКЛ ПОКА
```

Получение результирующей таблицы оператора INNER JOIN при помощи процессора обработки структур осуществляется в два этапа:

- пересечение индексов (первичных ключей) объединяемых структур данных;
- получение результирующих данных из структур в составе запроса на основе пересечения индексов.

Оператор INNER JOIN может быть реализован и с использованием последовательности операторов SEARCH в составе циклов, без применения операции объединения AND. В этом случае большое число команд SEARCH привело бы к замедлению выполнения запроса в сравнении с вариантом, включающим команду AND. Меньшая вычислительная сложность версии алгоритма с командой AND, аппаратно поддерживаемой процессором обработки структур МКОД-системы, послужила основной причиной выбора AND-варианта алгоритма. В итоге, выполнение команды AND сужает область дальнейшей обработки структур данных в МКОД-системе.

В рамках второго этапа выполнения оператора INNER JOIN процессором обработки структур, происходит последовательное определение тех пользовательских ID, которые входят в состав полученных по результатам операции AND групп пользователей, а также ID тех групп, которые включают полученные по результатам операции AND группы пользователей. После этого каждому пользователю ставится в соответствие от одной до нескольких групп, включающих его группу. Результаты записываются при помощи команды INSERT в структуру RESULT.

4.2. Оператор LEFT JOIN и LEFT OUTER JOIN

Для повышения скорости выполнения обработки запроса, содержащего оператор LEFT JOIN, необходимо добавить третью структуру данных (приведена в таблице 5). В рамках данной структуры ключом USR_GRP_REAL.KEY является идентификатор пользователя user_id, тогда как в качестве данных USR_GRP_REAL.DATA используются идентификаторы групп parent_group_id, в которые входят пользователи с данными идентификаторами user_id.

Таблица 5. Информационная модель для представления исходной таблицы user_in_group_membership

USR_GRP_REAL.KEY	USR_GRP_REAL.DATA
1	3
1,1	8
1,2	3
…	…
n	k
n,1	n_1
n,2	n_2
…	…
n,k	n_k

Оператор SQL LEFT JOIN для выполнения в МКОД-системе реализуется на основе оператора INNER JOIN, реализация которого для МКОД системы была приведена выше. В качестве «левой» структуры для оператора будем использовать USR_GRP, а в качестве «правой» - GRP_PARGRP. Псевдокод алгоритма для оператора LEFT JOIN в системе МКОД приведён ниже.

Алгоритм 2. Алгоритм выполнения запроса LEFT JOIN / псевдокод алгоритма в конвейерной системе

```
RESULT ← ∅
USRLIST ← ∅
in_join_struct ← INNERJOIN(USR_GRP, GRP_PARGRP)      □ выполнение INNER JOIN в МКОД варианте над исходными структурами
n ← POWER(USR_GRP)
users ← ∅
ЦИКЛ i = 1 to n                                      □ получаем множество ID пользователей из структуры USR_GRP
    userscnt ← SEARCH(USR_GRP, [i, 0])
    k ← userscnt.DATA
    ЦИКЛ j = 1 to k
      user ← SEARCH(USR_GRP, [i, j])
      users ← OR(users, user.DATA)
    ВСЕ ЦИКЛ
ВСЕ ЦИКЛ
remainder ← NOT(users, in_join_struct.KEY)           □ Получаем те пользовательские идентификаторы, которые не вошли в пересечение INNER JOIN
group_ids ← ∅
n ← POWER(remainder)
ЦИКЛ i = 1 to n                                      □ По аналогии с INNER JOIN строим таблицу user_id - parent_group_id
    usrgroups ← SEARCH(USR_GRP_REAL, [remainder[i], 0])
    k ← usrgroups.DATA
    ЦИКЛ j = 1 to k
group_id ← SEARCH(USR_GRP_REAL, [remainder[i], j]).DATA
      pargroups ← SEARCH(GRP_PARGRP, [group_id, 0])  □ Если значение на находится, то возвращается NULL
      m ← pargroups.DATA
      INSERT(RESULT, [remainder[i],0], m)
      ЦИКЛ j = 1 to m
        pargroup ← SEARCH(GRP_PARGRP, [group_id, j]).DATA
        INSERT(RESULT, [remainder[i], j], pargroup)
      ВСЕ ЦИКЛ
    ВСЕ ЦИКЛ
ВСЕ ЦИКЛ
RESULT ← OR(RESULT, in_join_struct)
```

Алгоритм 3. Алгоритм выполнения запроса LEFT OUTER JOIN / псевдокод алгоритма в конвейерной системе

```
RESULT ← ∅
USRLIST ← ∅
in_join_struct ← INNERJOIN(USR_GRP, GRP_PARGRP)   □ выполнение INNER JOIN в МКОД варианте над исходными структурами
n ← POWER(USR_GRP)
users ← ∅

ЦИКЛ i = 1 to n                                   □ получаем множество ID пользователей из структуры USR_GRP
    userscnt ← SEARCH(USR_GRP, [i, 0])

    k ← userscnt.DATA
    ЦИКЛ j = 1 to k
      user ← SEARCH(USR_GRP, [i, j])
      users ← OR(users, user.DATA)
    ВСЕ ЦИКЛ
ВСЕ ЦИКЛ
remainder ← NOT(users, in_join_struct.KEY)        □ Операция дополнения над индексами – получаем те пользовательские идентификаторы, которые не вошли в пересечение INNER JOIN
group_ids ← ∅
n ← POWER(remainder)

ЦИКЛ i = 1 to n                                   □ Для каждого пользовательского идентификатора получаем ID групп, в которые он входит, а потом ID групп, в которые входят эти группы, т.е. по аналогии с INNER JOIN строим таблицу user_id - parent_group_id
    usrgroups ← SEARCH(USR_GRP_REAL, [remainder[i], 0])
    k ← usrgroups.DATA
    ЦИКЛ j = 1 to k
      group_id ← SEARCH(USR_GRP_REAL, [remainder[i], j]).DATA
      pargroups ← SEARCH(GRP_PARGRP, [group_id, 0])
      m ← pargroups.DATA
      INSERT(RESULT, [remainder[i],0], m)
      ЦИКЛ j = 1 to m
        pargroup ← SEARCH(GRP_PARGRP, [group_id, j]).DATA
        INSERT(RESULT, [remainder[i], j], pargroup)
      ВСЕ ЦИКЛ
    ВСЕ ЦИКЛ
ВСЕ ЦИКЛ
```

Псевдокод алгоритма оператора LEFT OUTER JOIN представляет собой алгоритм оператора LEFT JOIN, но без последней строки, отвечающей за объединение множеств. Реализация алгоритма SQL LEFT JOIN включает в себя выполнение оператора SQL INNER JOIN, а также последовательность действий по поиску записей данных, не входящих в результат работы INNER JOIN, и объединению их с результатом работы оператора INNER JOIN.

Поскольку операторы RIGHT JOIN и RIGHT OUTER JOIN задаются симметрично операторам LEFT JOIN и LEFT OUTER JOIN соответственно, то их псевдокод приводить не будем для экономии места и сразу перейдём к оператору SQL FULL OUTER JOIN. Оператор SQL FULL JOIN в рамках работы рассматриваться не будет в связи с его относительно узким применением при реализации сервисов MDM-платформы.

4.3. Оператор FULL OUTER JOIN

Реализация оператора FULL OUTER JOIN становится весьма простой при наличии реализованных операторов LEFT OUTER JOIN и RIGHT OUTER JOIN. Псевдокод алгоритма для оператора FULL OUTER JOIN в МКОД-системе приведён ниже.

Алгоритм 4 Алгоритм выполнения запроса FULL OUTER JOIN / псевдокод алгоритма в конвейерной системе

1.	$RESULT \leftarrow \emptyset$	
2.	$RES1 \leftarrow LEFT.OUTER.JOIN(user\_in\_group\_membership, group\_in\_group\_membership)$	□ Получение левого внешнего соединения
3.	$RES2 \leftarrow RIGHT.OUTER.JOIN(user\_in\_group\_membership, group\_in\_group\_membership)$	□ Получение правого внешнего соединения
4.	$RESULT \leftarrow OR(RES1, RES2)$	□ Объединение результатов для получения внешнего соединения

Имеющаяся реализация операторов LEFT OUTER JOIN и RIGHT OUTER JOIN позволила заметно сократить псевдокод алгоритма, хотя в рамках выполнения операторов LEFT OUTER JOIN и RIGHT OUTER JOIN выполняются те же команды, что и в приведённом псевдокоде указанных операторов. Таким образом, в данном случае речь идёт исключительно об упрощении записи псевдокода алгоритма.

4.4. Реализация эффектов ключевых слов UNION, INTERSECT и EXCEPT на результаты SQL-запросов

Для корректной реализации объединения результатов двух и более SQL-запросов с использованием ключевого слова UNION, объединяемые таблицы результатов должны быть разделены на столбцы, каждый из которых входит в состав соответствующей структуры с добавленным ключевым столбцом, содержащим конкатенацию содержимого всех полей соответствующей записи исходной таблицы. Если поля значений оказываются для объединения слишком длинными, то для них может рассчитываться хэш-функция. Описанная схема подготовки данных приведена на рисунке 7. Пример результата использования данной схемы представлен в таблице 6.

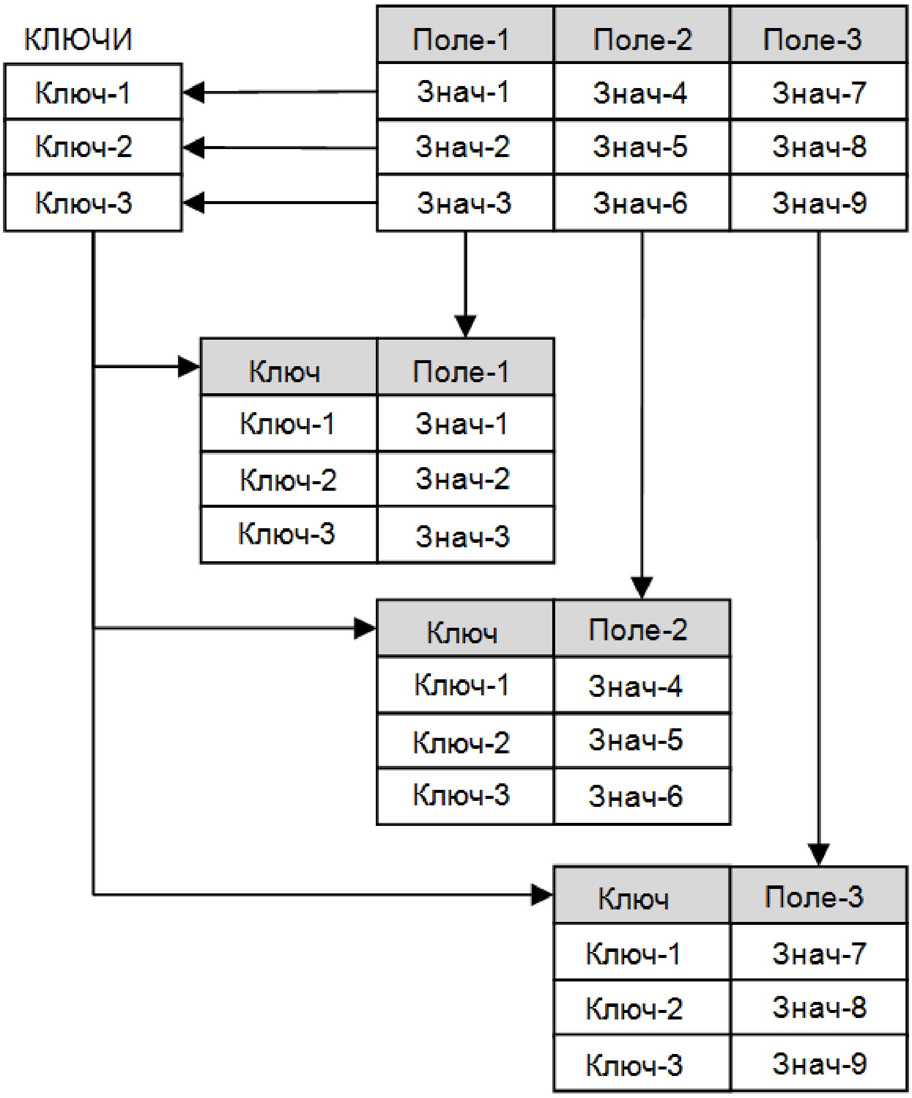

Рис. 7. Схема подготовки результата выполнения запроса для обработки в МКОД-системе при помощи ключевых слов SQL UNION, INTERSECT, EXCEPT

Таблица 6. Пример информационной модели для представления части исходной таблицы-результата SQL-запроса

A1.KEY	**A1.DATA**
ИВАН12М	ИВАН
ПЕТР14М	ПЕТР
СОФИЯ11Ж	СОФИЯ
…	…

Псевдокод алгоритма, реализующий функционал объединения таблиц-результатов запросов, представленных ранее, приведён ниже. Простота приведённого псевдокода обусловлена высокой аппаратной поддержкой операций над множествами и выбором корректного представления структур данных в МКОД-системе.

Алгоритм 5. Алгоритм выполнения объединения результатов SQL-запросов / псевдокод алгоритма в конвейерной системе

1. $RESULT[1..n] \leftarrow \emptyset$	□ Массив множеств-результатов
2. ЦИКЛ $i = 1\ to\ n$	□ Цикл по всем подтаблицам результатов (число подтаблиц совпадает у каждого из объединяемых результатов запросов)
3. $\quad RESULT[i] \leftarrow OR(A_i, B_i)$	□ Часть результата получается объединением частей результатов выполнения SQL-запросов
4. ВСЕ ЦИКЛ	

Аналогичным образом реализуются пересечение и дополнение результатов SQL-запросов, но, соответственно, с использованием аппаратно поддерживаемых МКОД-системой команд AND и NOT.

5. Тестирование выполнения SQL-запросов

Для проведения тестирования были выбраны SQL-операторы INNER JOIN, LEFT JOIN и LEFT OUTER JOIN. Было определено, что в рамках разработанной MDM-платформы данные операторы используются чаще всего. В рамках проведённых тестов каждый запрос был представлен в двух видах: стандартный код на языке С и код на языке С с командами процессора обработки структур. Выполнение SQL-запросов на языке С осуществлялось на компьютере с микропроцессором Intel Core 2 Duo (тактовая частота ядра – 2,6 ГГц) под управлением операционной системы Windows 7 (Максимальная). Выполнение SQL-запросов на языке С с командами процессора обработки структур осуществлялось на плате Xilinx University Program Virtex2Pro с ПЛИС Xilinx Virtex-II Pro XC2VP30 (тактовая частота – 100 МГц). В целях объективности проведённого эксперимента, было принято решение сравнивать обработку SQL-запросов устройствами при помощи определения числа тактов, затрачиваемых на выполнение операций (ввиду большой разницы в тактовой частоте). В таблице 7 представлены результаты тестов и суммарное число 32-разрядных ключей в двух таблицах. Для INNER JOIN размер обеих таблиц был выбран одинаковым, причём все данные в первой таблице являлись ключами для второй таблицы (худший случай для INNER JOIN), тогда как для LEFT JOIN и LEFT OUTER JOIN размер второй таблицы в два раза превышал размер первой, чтобы включить в первую таблицу данные, которые не используются в качестве ключей во второй, для их попадания в результат выполнения данных SQL-операторов.

Таблица 7. Результаты тестирования выполнения SQL-запросов ЭВМ МКОД (режим сопроцессора) и Intel Core 2 Duo

INNER JOIN			LEFT JOIN			LEFT OUTER JOIN		
Число ключей	**Число тактов (МКОД)**	**Число тактов (Intel)**	**Число ключей**	**Число тактов (МКОД)**	**Число тактов (Intel)**	**Число ключей**	**Число тактов (МКОД)**	**Число тактов (Intel)**
250000	361300742	1392777776	**375000**	562030776	801647132	**375000**	200731572	159684356
500000	722896884	2224145118	**750000**	851873415	1735541067	**750000**	401196036	328466788
750000	1084141196	2232938680	**1125000**	1279310511	2529629960	**1125000**	602626476	524374814
1000000	1447395492	2738455642	**1500000**	1705779783	3522943008	**1500000**	803469396	665010946
1250000	1809815372	3436790461	**1875000**	2133459999	4346198727	**1875000**	1004999884	891059000
1500000	2169748084	4262121812	**2250000**	2560793063	5226716170	**2250000**	1206352404	1025522069
1750000	2533487468	4947885280	**2625000**	2988639471	6108675456	**2625000**	1407932348	1178954554
2000000	2896582644	5660499468	**3000000**	3413990439	7028397883	**3000000**	1608189188	1808142312
2250000	3258727292	6453981651	**3375000**	3840350287	8106978319	**3375000**	1809051292	1588031029
2500000	3615737012	7793893408	**3750000**	4271066439	8865360933	**3750000**	2012109060	1758622931
2750000	3983855548	8083882042	**4125000**	4694182687	9919050637	**4125000**	2211248348	1953811197

В таблице 8 приведены значения «выигрыша» по числу тактов ЭВМ МКОД в сравнении с Intel Core 2 Duo. Сравнительные графики для числа тактов обеих конфигураций для всех протестированных SQL-запросов представлены на рисунках 8-10.

Таблица 8. Выигрыш по тактам, обеспечиваемый обработкой SQL-запросов ЭВМ МКОД (режим сопроцессора) по сравнению с Intel Core 2 Duo

№	INNER JOIN	LEFT JOIN	LEFT OUTER JOIN
1	3,85	1,43	0,80
2	3,08	2,04	0,82
3	2,06	1,98	0,87
4	1,89	2,07	0,83
5	1,90	2,04	0,89
6	1,96	2,04	0,85
7	1,95	2,04	0,84
8	1,95	2,06	1,12
9	1,98	2,11	0,88
10	2,16	2,08	0,87
11	2,03	2,11	0,88

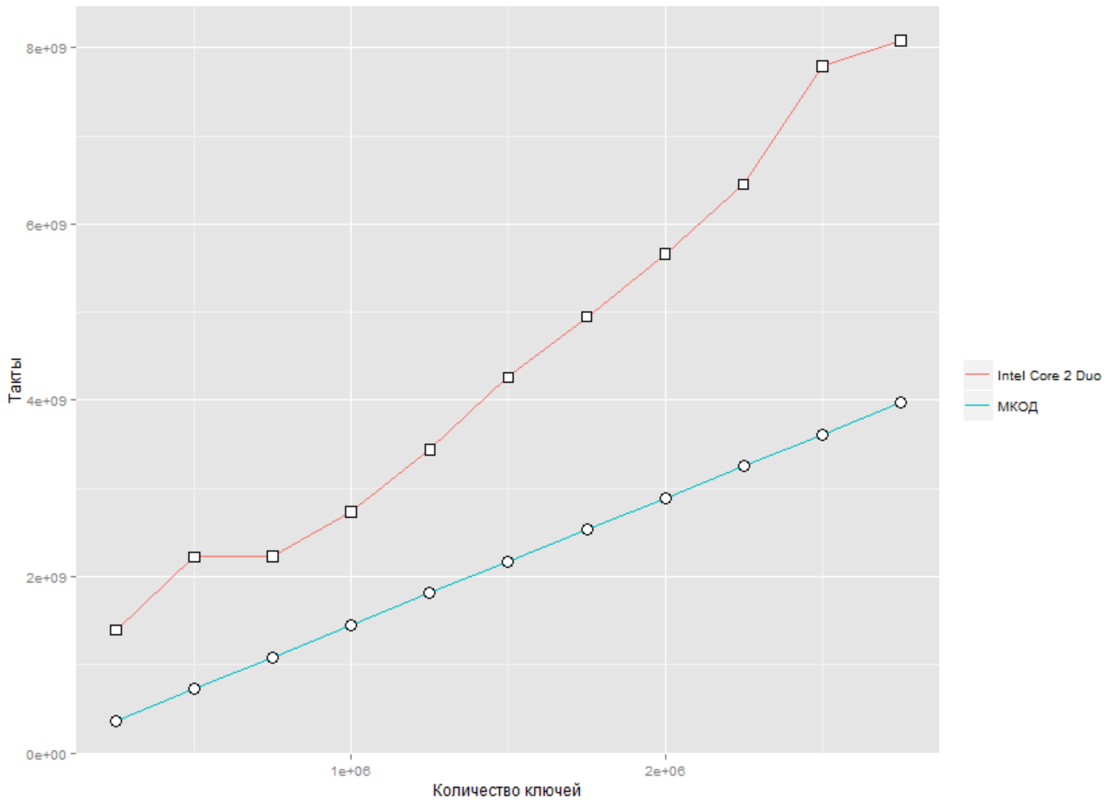

Рис. 8. Сравнение тактов для SQL-запроса SELECT с оператором INNER JOIN

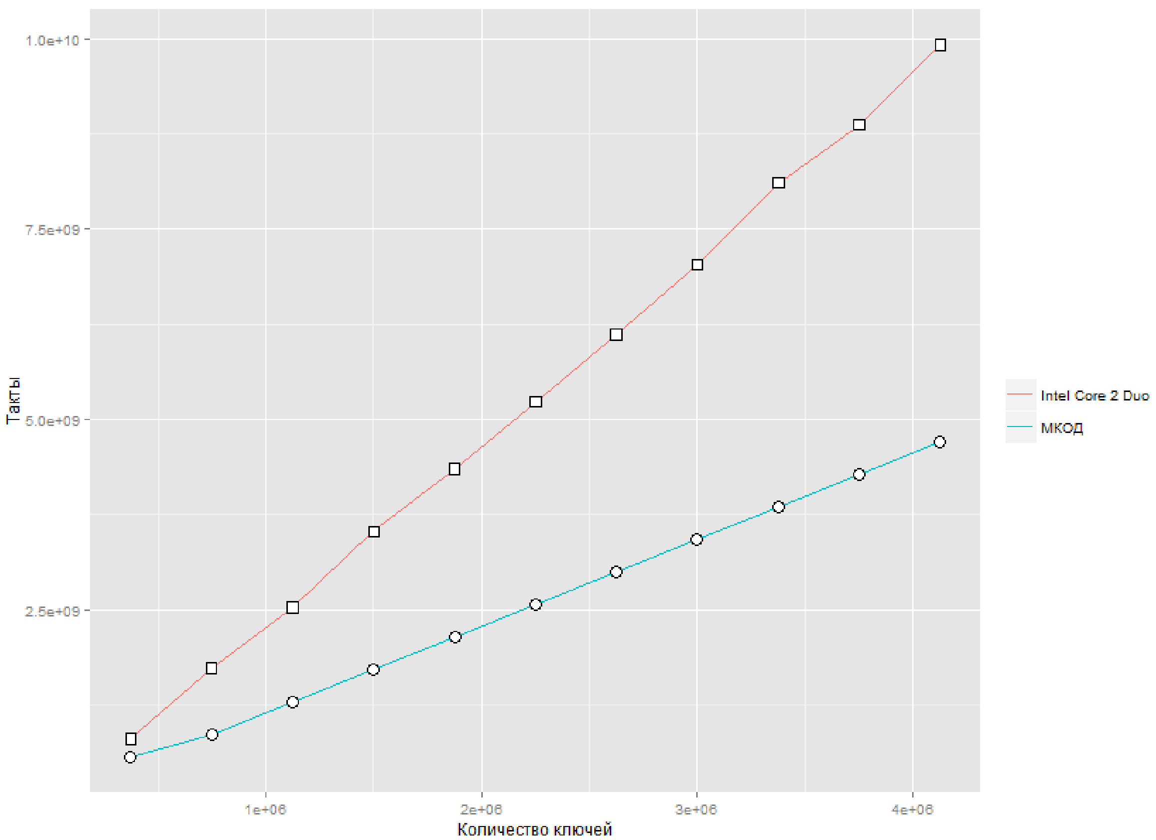

Рис. 9. Сравнение тактов для SQL-запроса SELECT с оператором LEFT JOIN

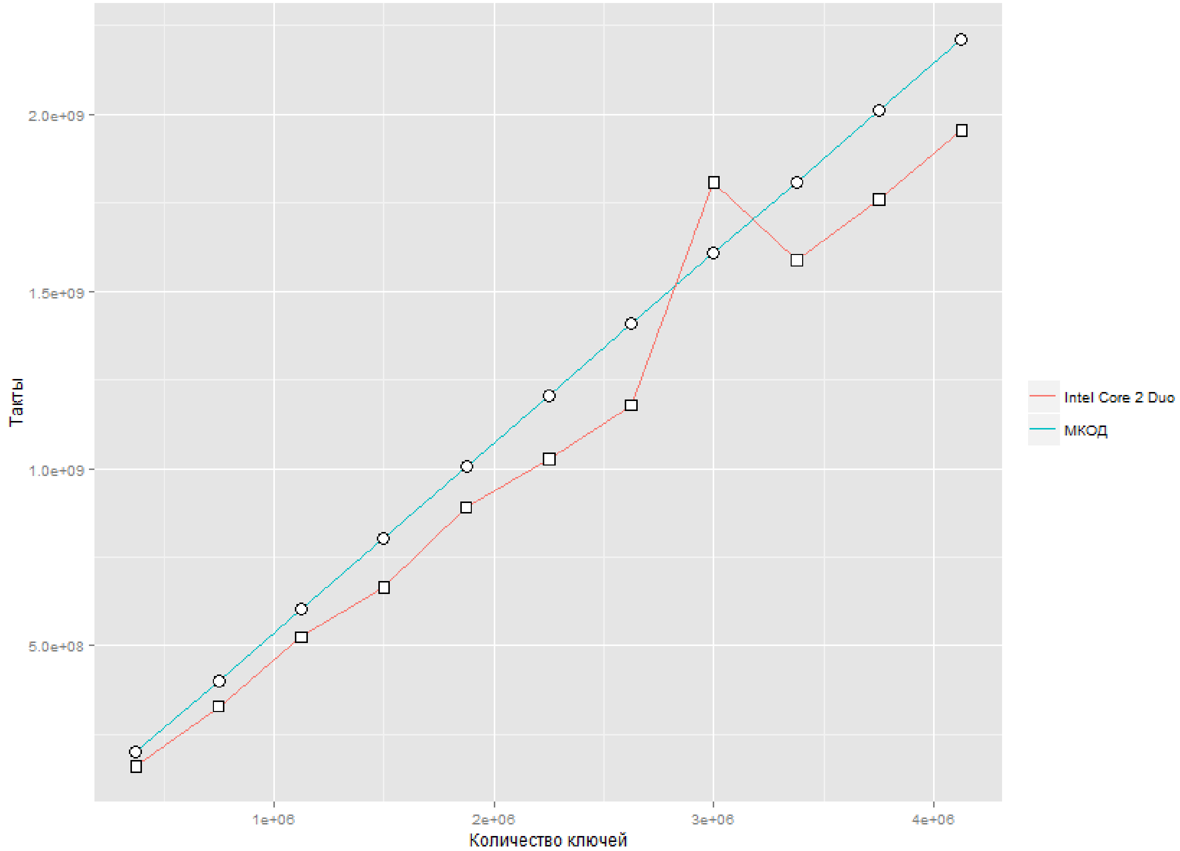

Рис. 10. Сравнение тактов для SQL-запроса SELECT с оператором LEFT OUTER JOIN

Можно заметить, что для случаев INNER JOIN и LEFT JOIN выигрыш по тактам составляет в среднем **2** раза, при этом для LEFT OUTER JOIN выигрыша не наблюдается. В связи с ограничениями по размеру памяти Xilinx University Program Virtex2Pro, на данном этапе разработки архитектуры ЭВМ МКОД не представляется возможным провести тестирование алгоритмов обработки SQL-запросов к базам данных промышленных размеров (более 1 Гб.).

6. Обсуждение – направления развития архитектуры МКОД для обработки SQL-запросов

Проведённые в рамках статьи исследования показали, что ряд изменённых для обработки на МКОД-системе алгоритмов SQL-запросов к базе данных может быть выполнен за меньшее число тактов, чем на других системах (выигрыш в два раза для INNER JOIN и LEFT JOIN). Для малого числа запросов к базе данных выигрыш от применения МКОД-системы неочевиден (например, случай LEFT OUTER JOIN), но при внедрении MDM-платформы на базе МКОД-решения в университете большое число запросов к базам данных MDM-платформы со стороны студентов, профессорско-преподавательского состава и гостей вуза будет обрабатываться быстрее за счёт аппаратной поддержки МКОД-решением операций над структурами данных. Тем не менее, МКОД-решение требует доработки для обеспечения полной аппаратной поддержки всех выполняемых в рамках MDM-парадигмы запросов.

В ходе проведённых исследований были выявлены следующие направления развития и доработки МКОД-системы:

- увеличение разрядности ключа в B+ дереве для реализации хранения и выполнения операций над большими массивами данных;
- реализация аппаратной поддержки операции декартова произведения над двумя и более множествами;
- увеличение размера памяти на борту платы для хранения большего объёма данных;
- реализация операции двойного среза на множестве. При наличии двух чисел $a,b \in R, a \leq b$ должна аппаратно поддерживаться операция получения множества B, состоящего из всех элементов некоторого множества A таких, что по своему значению они заключены в отрезке $[a;b]$, то есть $\forall c \in B: c \in A, c \geq a, c \leq b$;
- реализация аппаратной поддержки перекодирования SQL-запроса «на лету», то есть реализация автоматического преобразования SQL-запроса в МКОД-программу (в режиме сопроцессора) непосредственно в МКОД-системе;
- реализация полной поддержки графовых БД МКОД-системой. МКОД-система гораздо лучше обрабатывает запросы к структурам данных типа граф, но миграция на графовую СУБД в университете, очевидно, повлечёт издержки.

Проработка выявленных направлений совершенствования МКОД-решения позволит обеспечить реализацию полного набора базовых операций над данными, необходимого для поддержки переноса сложных информационных систем на МКОД-платформу. Как было показано в статье, осуществление подобного переноса возможно и целесообразно для информационных систем, значительная часть операций обработки данных в которых связана с осуществлением доступа к базам данных.

Заключение

В рамках настоящей статьи было подробно рассмотрено одно из возможных применений системы со многими потоками команд и одним потоком данных (МКОД). Применение МКОД-системы для обеспечения аппаратного ускорения выполнения запросов к базам данных позволит заметно повысить производительности MDM-платформы в целом, а также снизить энергопотребление технологического слоя MDM-платформы. Таким образом, материал данной статьи раскрывает возможности прикладного применения МКОД-системы в рамках бизнес-проектов. Безусловно, применение МКОД как ускорителя запросов к базам данных является более широким, чем MDM-платформы. Развитие и адаптация МКОД-решения в рамках выявленных направлений позволит существенно расширить набор решаемых МКОД прикладных задач в области получения и обработки данных при помощи запросов.

Помимо детального описания конкретного случая применения МКОД-решения в MDM-платформе, в статье основное внимание сосредоточено на формировании базового набора алгоритмов SQL-запросов для конвейерного режима МКОД-системы. Научная новизна настоящей статьи заключается в формировании набора аппаратно-поддерживаемых алгоритмов обработки SQL-запросов в МКОД-системе. В ходе формирования алгоритмов основных SQL-запросов, была выявлена потребность в доработке аппаратного решения МКОД-системы для обеспечения поддержки выполнения более сложных запросов к базам данных (например, включающих оператор CROSS JOIN). Представленные алгоритмы SQL-запросов для МКОД-системы были протестированы на прототипе МКОД-системы (ПЛИС Xilinx Virtex-II Pro XC2VP30) с использованием структур данных, полученных в ходе апробации MDM-платформы на базе университета. Показанные алгоритмами SQL-операторов INNER JOIN и LEFT JOIN результаты на ПЛИС по тактам оказались в среднем в два раза лучше, чем на Intel Core 2 Duo. Реализация LEFT OUTER JOIN показала лучший результат по тактам для Intel Core 2 Duo, чем для ПЛИС с МКОД-системой. В свете полученного результата для оператора LEFT OUTER JOIN необходимо расширение памяти на борту ПЛИС с МКОД-системой для продолжения тестирования на больших наборах данных, где теоретически МКОД-система может показать больший выигрыш по тактам.

Среди дальнейших направлений исследований и разработок в части совершенствования МКОД-системы следует выделить следующие:

- аппаратная реализация более широкого набора операций над структурами данных (множествами), которые требуются для реализации аппаратной поддержки сложных SQL-запросов к базам данных;
- исследование возможностей оптимизации представления таблиц данных в МКОД-системе;
- формирование библиотеки готовых МКОД-программ широкого назначения (поиск в глубину и ширину, алгоритмы на графах и др.);
- разработка программного и аппаратного решений, позволяющих осуществлять преобразование последовательных версий алгоритмов в некотором универсальном формате в МКОД-версию с параллельной обработкой структур данных и содержимого структур;
- исследование возможностей аппаратной поддержки преобразования SQL-запросов в формат МКОД «на лету».

Работа в указанных направлениях позволит получить законченное МКОД-решение, которое может быть использовано в коммерческих аппаратно-программных комплексах, специализирующихся на решении некоторого класса задач (например, связанных с обработкой SQL-запросов к базам данных).

Работы проводятся при финансовой поддержке государства в лице Министерства образования и науки Российской Федерации (государственный контракт № 02.G25.31.0024 от 12.02.2013).

Список литературы

1. Thomson G. BYOD: enabling the chaos // Network Security. 2012. Vol. 2012, is. 2. P. 5-8. DOI: 10.1016/S1353-4858(12)70013-2
2. Chiu Kin Fung. Adoption of mobile learning in schools: impact of changes in teacher values // Proc. of The 2015 International Mobile Learning Festival (IMLF2015), Hong Kong, China, 21-23 May 2015. P. 437-456.
3. Mouzaa C., Barrett-Greenly T. Bridging the app gap: An examination of a professional development initiative on mobile learning in urban schools // Computers & Education. 2015. Vol. 88. P. 1-14. DOI: 10.1016/j.compedu.2015.04.009
4. Becher A., Bauer F., Ziener D., Teich J. Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration // Proc. of the 24th International Conference on Field Programmable Logic and Applications (FPL). IEEE Publ., 2014. P. 1-8. DOI: 10.1109/FPL.2014.6927502
5. Castellana V.G., Minutoli M., Morari F., Tumeo A., Lattuada M., Ferrandi F. High Level Synthesis of RDF Queries for Graph Analytics // Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '15). IEEE Press, Piscataway, NJ, USA, 2015. P. 323-330.
6. Sukhwani B., Thoennes M., Min H., Dube P., Brezzo B., Asaad S., Dillenberger D. A Hardware/Software Approach for Database Query Acceleration with FPGAs // International Journal of Parallel Programming. 2015. Vol. 43, is. 6. P. 1129-1159. DOI: 10.1007/s10766-014-0327-4
7. Hsiue K.D. FPGA-based hardware acceleration for a key-value store database: M.Eng.Thesis. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2014. Available at: http://hdl.handle.net/1721.1/91829 (accessed 01.11.2015).
8. Попов А.Ю. Электронная вычислительная машина с многими потоками команд и одним потоком данных: пат. 71016 РФ. 2008. Бюл. № 5. 1 с.
9. Samochadin A., Timofeev D., Maslov M. Architecture of a Platform for Building Context-Aware Educational Mobile Services // Proceedings of the 11th International Conference on Engineering Education (EDUCATION '15), Salerno, Italy, June 27-29, 2015. P. 36-40.
10. Попов А.Ю. Исследование производительности процессора обработки структур в системе с многими потоками команд и одним потоком данных // Инженерный журнал: наука и инновации. 2013. № 11. DOI: 10.18698/2308-6033-2013-11-1048
11. Попов А.Ю. О реализации алгоритма Форда-Фалкерсона в вычислительной системе с многими потоками команд и одним потоком данных // Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2014. № 9. С. 162-180. DOI: 10.7463/0914.0726416
12. Попов А.Ю. Реализация алгоритмов обхода графов в вычислительной системе с многими потоками команд и одним потоком данных // Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. № 10. С. 453-472. DOI: 10.7463/1015.0820736

Science & Education of the Bauman MSTU. Electronic journal. ISSN 1994-0408

Science and Education of the Bauman MSTU, 2015, no. 12, pp. 165–196.

DOI: **10.7463/1215.0827960**

Received: 16.11.2015
Revised: 30.11.2015

Hardware Acceleration of SQL-Queries Processing in MDM-Systems Based on MISD-Solution

V.E. Podol'skii[1,*], A.V. Samochadin[2], S.S. Koloskov[1]

[*]v.e.podolskiy@gmail.com

[1]Bauman Moscow State Technical University, Moscow, Russia
[2]Peter the Great St.Petersburg Polytechnic University, St.Petersburg, Russia

Keywords: mobile device management, MDM, MISD, SQL, data structures processing, structures processing unit, BYOD

In this article we examine the possibility of hardware support for functions of mobile device management platform (MDM-platform) using a Multiple Instructions and Single Data stream computer system, developed within the framework of the project in Bauman Moscow State Technical University. At the universities the MDM-platform is used to provide various mobile services for the faculty, students and administration to facilitate the learning process: a mobile schedule, document sharing, text messages, and other interactive activities. Most of these services are provided by the extensive use of data stored in MDM-platform databases. When accessing the databases SQL- queries are commonly used. These queries comprise operators of SQL-language that are based on mathematical sets theory. Hardware support for operations on sets is implemented in Multiple Instructions and Single Data stream computer system (MISD System). This allows performance improvement of algorithms and operations on sets. Thus, the hardware support for the processing of SQL-queries in MISD system allows us to benefit from the implementation of SQL-queries in the MISD paradigm.

The scientific novelty of the work lies in the fact that it is the first time a set of algorithms for basic SQL statements has been presented in a format supported by MISD system. In addition, for the first time operators INNER JOIN, LEFT JOIN and LEFT OUTER JOIN have been implemented for MISD system and tested for it (testing was done for FPGA Xilinx Virtex-II Pro XC2VP30 implementation of MISD system). The practical significance of the work lies in the fact that the results of the study will be used in the project "Development of the Russian analogue of the system software for centralized management of personal devices and platforms in enterprise networks" of the St. Petersburg Polytechnic University (with the financial support of the state represented by the Ministry of Education and Science of the Russian Federation).

In this study, we have conducted the testing on the basis of data structures obtained during the approbation of MDM-platform in SPbPU. Test results have shown that on average MISD system provides a two-fold gain for SQL-operators such as INNER JOIN and LEFT JOIN. However, it was found that MISD system does not give gain in cycles for SQL-operator LEFT OUTER JOIN. A comparison was carried out with the processing of requests on Intel Core 2 Duo.

The results of this work are areas of further development and improvement of MISD system, including the area of hardware support for new operations.

References

1. Thomson G. BYOD: enabling the chaos. *Network Security*, 2012, vol. 2012, is. 2, pp. 5-8. DOI: 10.1016/S1353-4858(12)70013-2
2. Chiu Kin Fung. Adoption of mobile learning in schools: impact of changes in teacher values. *Proc. of The 2015 International Mobile Learning Festival (IMLF2015)*, Hong Kong, China, 21-23 May 2015, pp. 437-456.
3. Mouzaa C., Barrett-Greenly T. Bridging the app gap: An examination of a professional development initiative on mobile learning in urban schools. *Computers and Education*, 2015, vol. 88, pp. 1-14. DOI: 10.1016/j.compedu.2015.04.009
4. Becher A., Bauer F., Ziener D., Teich J. Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration. *Proc. of the 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE Publ., 2014, pp. 1-8. DOI: 10.1109/FPL.2014.6927502
5. Castellana V.G., Minutoli M., Morari F., Tumeo A., Lattuada M., Ferrandi F. High Level Synthesis of RDF Queries for Graph Analytics. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '15).* IEEE Press, Piscataway, NJ, USA, 2015, pp. 323-330.
6. Sukhwani B., Thoennes M., Min H., Dube P., Brezzo B., Asaad S., Dillenberger D. A Hardware/Software Approach for Database Query Acceleration with FPGAs. *International Journal of Parallel Programming*, 2015, vol. 43, is. 6, pp. 1129-1159. DOI: 10.1007/s10766-014-0327-4
7. Hsiue K.D. *FPGA-based hardware acceleration for a key-value store database. M.Eng.Thesis*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2014. Available at: http://hdl.handle.net/1721.1/91829 , accessed 01.11.2015.
8. Popov A.Yu. Elektronnaia vychislitel'naia mashina s mnogimi potokami komand i odnim potokom dannykh [Electronic computer with multiple instruction streams and single data stream]. Patent RF, no. 71016, 2008. (in Russian).
9. Samochadin A., Timofeev D., Maslov M. Architecture of a Platform for Building Context-Aware Educational Mobile Services. *Proceedings of the 11th International Conference on Engineering Education (EDUCATION '15)*, Salerno, Italy, June 27-29, 2015, pp. 36-40.

10. Popov A.Yu. The study of the structure processor performance in the computer system with multiple-instruction streams and single-data stream. *Inzhenernyy zhurnal: nauka i innovatsii = Engineering Journal: Science and Innovation*, 2013, no. 11. DOI: 10.18698/2308-6033-2013-11-1048 (in Russian).
11. Popov A.Yu. On the implementation of the Ford-Fulkerson algorithm on the Multiple Instruction and Single Data computer system. *Nauka i obrazovanie MGTU im. N.E. Baumana = Science and Education of the Bauman MSTU*, 2014, no. 9, pp. 162-180. DOI: 10.7463/0914.0726416 (in Russian).
12. Popov A.Yu. The implementation of graph traversal algorithms on the Multiple Instruction and Single Data stream computer system. *Nauka i obrazovanie MGTU im. N.E. Baumana = Science and Education of the Bauman MSTU*, 2015, no. 10, pp. 453-472. DOI: 10.7463/1015.0820736 (in Russian).