Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Смоленский государственный университет»

На правах рукописи

Мунерман Виктор Иосифович

АЛГЕБРАИЧЕСКИЕ МОДЕЛИ И МЕТОДЫ ДЛЯ РАЗРАБОТКИ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ МАССОВОЙ ОБРАБОТКИ ДАННЫХ

2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация на соискание учёной степени
доктора технических наук

Смоленск – 2024

Оглавление

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ ... 5
ВВЕДЕНИЕ ... 6
Глава 1. МАССОВАЯ ОБРАБОТКА ДАННЫХ: ОПРЕДЕЛЕНИЕ, МЕТОДЫ ОПИСАНИЯ, АНАЛИЗ ПРОБЛЕМ ... 17
1.1. Понятие массовой обработки данных ... 17
1.2. Связь моделей МОД с архитектурами программно-аппаратных комплексов ... 29
1.2.1. Логически последовательный метод доступа ... 29
1.2.2. Архитектуры вычислительных комплексов для реализации логически последовательного метода доступа ... 32
1.3. Проблемы оптимизации процессов массовой обработки данных ... 40
1.4. Требования к моделям массовой обработки данных ... 46
1.5. Заключительные замечания к главе 1 ... 51
Глава 2. АЛГЕБРАИЧЕСКИЕ МОДЕЛИ ДАННЫХ ДЛЯ ПОСТРОЕНИЯ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ ... 53
2.1. Основные алгебраические понятия ... 53
2.1.1. Универсальные алгебры и алгебраические системы ... 53
2.1.2. Интуитивный подход к объектно-ориентированному моделированию, проектированию и программированию ... 56
2.1.3. Алгебраический (формальный) подход к объектно-ориентированному моделированию, проектированию и программированию ... 59
2.1.4. Объектно-ориентированный подход к разработке моделей данных ... 62
2.2. Файловая (теоретико-множественная) модель данных ... 68
2.2.1. Анализ определений файла ... 68
2.2.2. Определение файла ... 70
2.2.3. Описание операций над файлами ... 73
2.3. Многомерно-матричная модель данных ... 78
2.3.1. Задачи многомерно-матричного представления данных ... 78
2.3.2. Алгебра многомерных матриц ... 80
2.4. Соответствие моделей данных ... 87
2.4.1. Соответствие теоретико-множественной и многомерно-матричной моделей ... 88
2.4.2. Соответствие промежуточных моделей данных высокоуровневым моделям ... 94
2.5. Аксиоматический подход к формализации моделей данных для МОД ... 97
2.5.1. Соответствие аксиоматического и алгебраического подходов к формализации МОД ... 98
2.5.2. Определение аксиоматической теории МОД ... 100
2.5.3. Интерпретация формул аксиоматической теории МОД ... 101
2.5.4. Аксиомы теории массовой обработки данных ... 102
2.5.5. Теоремы аксиоматической теории МОД ... 105
2.5.6. Эквивалентность моделей МОД ... 107

2.5.7. Доказательство эквивалентности моделей МОД по операциям слияния 108
2.6. Заключительные замечания к главе 2 115
Глава 3. МЕТОДЫ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ МАССОВОЙ ОБРАБОТКИ ДАННЫХ 117
3.1. Проблемы повышения эффективности обработки данных 117
3.2. Синтез и оптимизация процесса МОД 122
3.2.1. Модель и метод построения процесса МОД 122
3.2.2. Формальное описание метода синтеза и оптимизации процесса МОД 125
3.2.3. Реализация метода синтеза и оптимизации процесса МОД 130
3.3. Выбор параллельных алгоритмов для реализации операций МОД 134
3.4. Алгоритм параллельного умножения многомерных матриц 136
3.4.1. Выбор алгоритма умножения многомерных матриц 136
3.4.2. Описание алгоритма умножения многомерных матриц 141
3.5. Алгоритм параллельной реализации операции слияния нестрого упорядоченных файлов 149
3.5.1. Анализ алгоритмов параллельной реализации операции слияния нестрого упорядоченных файлов 149
3.5.2. Организация данных для параллельной реализации операции слияния нестрого упорядоченных файлов 151
3.5.3. Параллельный алгоритм реализации операции слияния нестрого упорядоченных файлов 154
3.5.3.1. Эвристический алгоритм распределения 155
3.6. Алгоритм параллельной реализации операции соединения в реляционной модели SQL ... 165
3.7. Стратегия повышения эффективности процессов МОД 170
3.8. Заключительные замечания к главе 3 171
Глава 4. АРХИТЕКТУРЫ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ ДЛЯ МАССОВОЙ ОБРАБОТКИ ДАННЫХ 173
4.1. Этапы построения программно-аппаратных комплексов 173
4.2. Архитектура программно-аппаратного комплекса для реализации многомерно-матричной модели данных 176
4.3. Архитектуры программно-аппаратных комплексов для реализации простых операций теоретико-множественной модели данных 179
4.3.1. Параллельная реализация внешней сортировки 179
4.3.2. Параллельная реализация операций выборки, слияния строго упорядоченных файлов и сечения 183
4.4. Параллельная реализация операции слияния нестрого упорядоченных файлов в теоретико-множественной и реляционной моделях данных 185
4.4.1. Параллельная реализация операции слияния нестрого упорядоченных файлов алгоритмом черпака 188
4.4.2. Параллельная реализация последовательности операций слияния нестрого упорядоченных файлов 192

4.4.3. Параллельная реализация операции слияния нестрого упорядоченных файлов с использованием ассоциативных вычислительных систем ... 195
4.5. Заключительные замечания к главе 4 ... 198
Глава 5. ЭКСПЕРИМЕНТАЛЬНЫЙ АНАЛИЗ ПРОГРАММНО-АППАРАТНЫХ РЕАЛИЗАЦИЙ МАССОВОЙ ОБРАБОТКИ ДАННЫХ ... 200
5.1. Анализ параллельной реализации операции умножения многомерных матриц ... 200
5.2. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов ... 202
5.3. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов средствами СУБД и SMP-архитектуры ... 202
5.4. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием MPP-архитектуры в облачной среде ... 206
5.5. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием SMP-архитектуры в облачной среде ... 211
5.6. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием SMP-архитектуры и многоядерных графических процессоров ... 213
5.7. Заключительные замечания к главе 5 ... 214
Глава 6. ПРИМЕНЕНИЕ АЛГЕБРАИЧЕСКОГО ПОДХОДА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ ... 216
6.1. Использование предложенного метода для решения задач о кратчайшем пути ... 216
6.1.1 Решение традиционной задачи ... 216
6.1.2. Решение задачи с одновременным построением пути ... 221
6.2. Использование предложенного метода для решения задачи вывода ассоциативных правил ... 229
6.3. Использование предложенного метода для решения задачи поиска изображений в базах данных ... 234
6.3.1. Архитектура программно-аппаратного комплекса для поиска изображений в базах данных ... 235
6.3.2. Параллельное сравнение ключей изображений ... 237
6.3.4. Реализация и анализ метода поиска изображений в базе данных ... 240
6.4. Реализация алгоритма шифрования Хилла на основе алгебры многомерных матриц ... 243
6.4.1. Краткое описание алгоритма шифрования Хилла ... 244
6.4.2. Дополнительные элементы алгебры многомерных матриц ... 245
6.5. Заключительные замечания к главе 6 ... 251
Заключение ... 253
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ... 255
Приложение 1. Патент на полезную модель RUS 82355 ... 280
Приложение 2. Патент № 2755568 ... 281
Приложение 3. Свидетельства о регистрации программы для ЭВМ ... 284
Приложение 4. Акт внедрения результатов диссертационной работы ... 285

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ

ASM автоматическая система управления (Oracle Automatic Storage Management) для Oracle Exadata
AVX – расширение системы команд x86 для микропроцессоров Intel и AMD (Advanced Vector Extensions)
CALS (Continuous Acquisition and Life cycle Support – непрерывная информационная поддержка поставок и жизненного цикла издели
CUDA – программно-аппаратная архитектура параллельных вычислений
DBRM – подсистемf управления ресурсами Oracle
DDL – язык управления данными
DML – язык манипулирования данными
FPGA – программируемая пользователем вентильная матрица
GPU – графический процессор
ISAM – индексно-последовательный метод доступа
MDX (Multidimensional Expressions) – SQL-подобный язык запросов, ориентированный на доступ к многомерным структурам данных.
MIMD – архитектура "множественный поток команд, множественный поток данных"
MISD – архитектура "множественный поток команд, одиночный поток данных"
MOLAP – многомерная интерактивная аналитическая обработка
MPP – массивно-параллельная архитектура
MSSQL – Microsoft Sql Server
NUMA – архитектура с неравномерной памятью
ODBC – программный интерфейс доступа к базам данных
OLAP online analytical processing, интерактивная аналитическая обработка)
SIMD – архитектура "одиночный поток команд, множественный поток данных"
SMP – архитектура симметричная многопроцессорность
SPMD – архитектура одна программа много данных
АСУ – автоматизированная система управления
АТД – абстрактный тип данных
БД – база данных
БНФ – форма Бэкуса-Наура
ВСАРР – вычислительная система с ассоциативным распределением ресурсов
ЛММ – логическая многомерная матрица
МОД – массовая обработка данных
ПЛИС – программируемая логическая интегральная схема
СУБД – система управления базами данных

ВВЕДЕНИЕ

Общая характеристика работы

В работе предлагаются и рассматриваются математическая и алгоритмическая поддержка массовой обработки данных на основе алгебраических моделей. Традиционно массовую обработку данных связывают с параллельными вычислениями и чаще всего определяют следующим образом: массовая параллельная обработка – способ параллельной обработки больших объемов данных большим числом процессоров.

Актуальность темы. В настоящее время массовую обработку данных связывают с направлением, получившим название Big data. Big data (большие данные) – общий термин, который обозначает вновь создающиеся структурированные, неструктурированные и полуструктурированные данные сверхбольших и постоянно возрастающих объемов. Загрузка неструктурированных и полуструктурированных данных в обычную (например, реляционную) базу данных и последующая обработка требуют слишком больших затрат ресурсов вычислительных комплексов. Поэтому работа посвящена рассмотрению распространенного класса массовой обработки – обработке высокоактивных структурированных данных. Высокая активность данных означает, что при выполнении операций над данными, в обработку включается их большая часть, близкая к ста процентам.

Большие данные (big data) не только неотъемлемая часть современного мира обработки данных, но и основная его часть. Комплекс сложных научно-технических проблем, связанных с решением задач на основе обработки больших данных, при переходе к информационному обществу не только сохраняет, но и усиливает свою актуальность. Об этом свидетельствуют интенсивные научные исследования в области баз данных, проводимые в России и за рубежом.

Требования к обработке больших данных, существенно влияют на технологию разработки программных и аппаратных вычислительных средств ее реализующих. Обработка больших данных необходима при решении сложных

вычислительных задач [1, 2, 3], информационно-логических задач, к которым относятся обработка транзакционных запросов и запросов, требующих больших рабочих нагрузок [4, 5], задачи в области искусственного интеллекта [6-9], такие как переобучение сверточных нейронных сетей [10-12]. Эти задачи обусловили необходимость использования параллельной обработки и обработки в основной (оперативной) памяти.

Важное значение в обработке больших данных имеют параллельные и распределенные базы данных, для которых создаются специализированные программно-аппаратные комплексы – машины баз данных [13, 14]. Большинство современных машин баз данных ориентированы на реализацию реляционных СУБД, которые недавно стали широко применяться для подготовки исходных данных в системах глубокого обучения [15-118]. При разработке параллельных вычислительных систем особую роль играет повышение эффективности реализации отдельных операций, имеющих большую вычислительную сложность, о чем свидетельствуют многие публикации, например, [19-23]. К этим операциям относятся такие как многомерное дискретное преобразование Фурье, свертки в процессе обучения нейронных сетей, операция Join в реляционных системах баз данных.

Современные аппаратные средства, такие как многопоточные процессоры архитектур подобных x86 и ARMv8, а также графические и тензорные процессоры (GPU, TPU), позволяют проектировать параллельные вычислительные системы, которые обеспечивают решение многих из перечисленных задач [24-33]. Важное направление основано на использовании современных программируемых логических интегральных схем (ПЛИС/FPGA). Относительная простота программирования [34, 35] позволяет использовать их для решения как вычислительных, так и информационно-логических задач, и создания центров обработки данных (Data Center) [36-40]. Особую роль современные ПЛИС играют в решении проблемы построения быстро реконфигурируемых вычислительных систем [41]. Так в машине баз данных IBM Netezza [42, 43], ПЛИС позволяет быстро перестраивать процессор обмена и подготовки данных S-Blade, превра-

щая его в специализированный процессор для выполнения конкретного запроса, полученного от случайного пользователя. Также для массовой обработки данных могут эффективно использоваться специализированные процессоры [44-48].

Рассмотренные важные задачи, требуют для своего решения построения программно-аппаратных комплексов, основное свойство которых заключается в достижении эффективного баланса программного и аппаратного обеспечения [49-51]. Эта эффективность достигается тогда, когда как сказано в [52 – Воеводин В.В. Вычислительная математика и структура алгоритмов. – М.: Изд-во МГУ, 2006. – 112 с. – ISBN 5-211-05310-9]: "Для всех конкретных параллельных вычислительных систем степень согласованности структуры алгоритмов с архитектурой систем играет самую важную роль в достижении наивысших скоростей". Формальная постановка этой проблемы предложена в [53 – Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. – Киев: Наукова думка, 1989. – 376 с. – ISBN 5-12-000499-7]: "Одним из основных источников задач прикладной теории алгоритмов является проблема оптимального перевода с одного языка на другой, которая может быть сформулирована следующим образом: существуют два алгоритмических языка и некоторый алгоритм, написанный на одном из них; требуется найти оптимальную по заданным критериям реализацию этого алгоритма на другом языке. В программировании обычно первым является некоторый язык программирования, ориентированный на тот или иной круг задач, а вторым – внутренний язык машины, на которой решаются данные задачи."

Поиск причин возникновения трудностей при решении задач на вычислительной технике параллельной архитектуры неизбежно приводит к выводу, что ***как истоки этих причин, так и пути их преодоления надо искать в математических знаниях об алгоритмах***".

Эти две посылки позволяют сделать заключение, состоящее в следующем. Алгебраическая система, в которой формализована решаемая задача, и модель вычислений (алгебраическая система, реализованная в системе команд

вычислительной системы или комплекса) должны в наибольшей степени соответствовать друг другу. Эти алгебраические системы должны быть, по крайней мере, гомоморфными, а в идеальном случае, когда достигается полное соответствие – изоморфными.

Для обработки больших данных традиционно используется массовая обработка данных (massively data processing, massively data computing) – способ параллельной обработки больших объемов данных большим числом процессоров. Для ее реализации проектируются и используются специализированное программно-аппаратные комплексы [54-57].

Область, в которой применение массовой обработки данных имеет важное значение – это обработка высокоактивных данных. Активность данных определяется отношением числа обращений к элементу данных (записи файла, элементу матрицы, строки таблицы) к общему числу обращений к информации (файлу, матрице, базе данных) в единицу времени.

Важным примером применения предложенных в диссертационной работе методов построения программно-аппаратных комплексов для массовой обработки высокоактивных данных служит упомянутое ранее реляционное глубокое обучение. В реляционной базе данных выполняются запросы на формирование обучающей и тестирующей выборок. Поскольку системы глубокого обучения базируются на алгебре тензоров – частном случае алгебры многомерных матриц [58, 59], то целесообразно для подготовки данных для систем глубокого обучения использовать многомерно-матричные машины баз данных [60].

В соответствии со сказанным в работа содержит подробное описание и развитие ***формализованного математического, а именно, алгебраического аппарата***, который обеспечит наилучшее соответствие ***характера данных*** и ***свойств вычислительных средств***, для реализации массовой обработки данных, что подтверждает ее актуальность.

Цель исследования. Целью диссертации является создание алгебраических моделей и методов синтеза программно-аппаратных комплексов массовой обработки высокоактивных данных путем выбора на их основе подходящих ар-

хитектур и способов построения вычислительных процедур.

Задачи исследования. Для достижения поставленной цели необходимо было решить следующие задачи.

1. Разработка алгебраического подхода к организации моделей данных и моделей вычислений, выработка системы требований к этому классу алгебраических моделей и проведение анализа универсальных алгебраических систем в большей мере отвечающих предъявляемым требованиям, разработка формальной алгебраической модели массовой обработки данных – теоретико-множественной файловой модели, и обоснование эффективности применения в качестве моделей данных и моделей вычислений алгебры многомерных матриц и реляционной алгебры.
2. Разработка системы методов доказательства соответствия предложенных алгебраических моделей с использованием алгебраических доказательств гомоморфизма или изоморфизма, и аксиоматических – доказательства эквивалентности теорий.
3. Разработка методов синтеза алгебраических моделей процессов массовой обработки данных на основе принципов динамического программирования.
4. Разработка методов и алгоритмов параллельной реализации операций массовой обработки данных, архитектур программно-аппаратных комплексов, реализующих эти операции и анализ приемлемости параллельной реализации процессов массовой обработки данных на предложенных архитектурах программно-аппаратных комплексов.
5. Разработка объектно- и предметно-ориентированных технологий, и средств методического и программно-технического обеспечения для массовой обработки высокоактивных структурированных данных
6. Реализация решения прикладных задач из различных предметных областей на основе предложенных методов массовой обработки данных.

Методы исследования. Проведенные в работе исследования базируются на математических моделях данных и вычислений, таких как алгебра многомерных матриц, реляционная алгебра, теория множеств, и используют методы

математического моделирования. Для решения поставленных задач применялись аппарат математического анализа, теории алгебраических систем и метаматематики, методы системного, модульного, функционального и объектно-ориентированного программирования, а также технология параллельной обработки данных на многопроцессорных программно-аппаратных комплексах.

Основные положения, выносимые на защиту

На защиту выносятся следующие новые научные результаты.

1. Предложена новая теоретико-множественная (файловая) модель массовой обработки данных. Дано определение файла, как фактор-множества множества однотипных записей по отношению эквивалентности, порожденному множеством ключей. На основе этого определения сделаны формальные определения основных операций массовой обработки данных с использованием алгебраического и объектно-ориентированного подхода, которые позволили формализовать как операции над структурами данных высокого уровня – файлами, так и над составляющими их элементами – записями (кортежами).
2. Разработана алгебраическая модель данных и вычислений на основе алгебры многомерных матриц. Доказано соответствие этой модели теоретико-множественной и реляционной моделям.
3. Разработана аксиоматическая теория массовой обработки данных. Показано, что запросы на обработку данных – есть теоремы аксиоматической теории. На основе теоремы об эквивалентности аксиоматических теорий доказано, что многомерно-матричная и реляционная модели данных соответствуют друг другу.
4. Разработано обобщение алгоритма оптимизации последовательного умножения матриц на умножение многомерных матриц. Разработан метод на основе динамического программирования для синтеза оптимизированного процесса массовой обработки данных. Предложена стратегия оптимизации таких процессов.

5. Предложен и исследован метод симметричного горизонтального распределения таблиц-операндов операции JOIN, для последующего параллельного выполнения этой операции над фрагментами таблиц-операндов.

6. Предложены архитектуры программно-аппаратных комплексов для параллельного выполнения процессов массовой обработки данных, формализованных в различных моделях данных: многомерно-матричной, теоретико-множественной (файловой), реляционной.

Научная новизна

Научная новизна работы заключается в следующем:

1. Предложен новый метод формализации моделей данных и моделей вычислений, основанный на универсальных многоосновных алгебраических системах и объектно-ориентированном подходе к проектированию и разработке программно-аппаратных комплексов для решения задач массовой обработки данных.

2. Предложена и разработана теоретико-множественная (файловая) алгебраическая система, которая используется для верификации соответствия известных и используемых на практике моделей данных и моделей вычислений.

3. Описана алгебра многомерных матриц и предложен метод – абстрактная алгебраическая машина, который позволяет использовать в качестве элементов многомерных матриц различные, как простые, так и структурные (кортежи), типы данных.

4. Разработаны алгебраический и аксиоматический методы подтверждения соответствия (изоморфизма или гомоморфизма) моделей данных и моделей вычислений и осуществлено подтверждение гомоморфизма, а для конкретных задач – изоморфизма, теоретико-множественной, многомерно-матричной и реляционной моделей на основе использования обоих методов.

5. Разработано обобщение алгоритма выбора последовательности операций умножения матриц методом динамического программирования для (□, □)-свернутого произведения многомерных матриц, показано, что синтез оптимизированного процесса МОД может быть реализован методом динамического

программирования. Приведен пример синтеза такого процесса.

6. На основе параллельной реализации операций: выбран и обобщен для параллельной реализации (λ, μ)-свернутого произведения многомерных матриц алгоритм Кэннона; для файловой модели разработан алгоритм операции слияния нестрого упорядоченных файлов на основе симметричного горизонтального распределения файлов-операндов; для реализации симметричного горизонтального распределения разработан оригинальный эвристический алгоритм; показано, что на основе симметричного горизонтального распределения может быть эффективно распараллелена реляционная операция Join.

7. Разработаны этапы построения программно-аппаратного комплекса для реализации МОД, архитектура для реализации многомерно-матричной модели данных, архитектуры для реализации простых (однопроходных) операций теоретико-множественной модели данных, архитектура для параллельной реализации операции слияния нестрого упорядоченных файлов в теоретико-множественной и реляционной моделях данных для различных алгоритмов, в том числе с использованием ассоциативных вычислительных систем.

Соответствие паспорту научной специальности 2.3.5. «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»

Полученные в диссертационной работе результаты соответствуют пунктам:

3 – Модели, методы, архитектуры, алгоритмы, языки и программные инструменты организации взаимодействия программ и программных систем.

8 – Модели и методы создания программ и программных систем для параллельной и распределенной обработки данных, языки и инструментальные средства параллельного программирования.

9 – Модели, методы, алгоритмы, облачные технологии и программная инфраструктура организации глобально распределенной обработки данных.

Теоретическая и практическая значимость

Теоретическая значимость диссертации состоит в том, что на основе раз-

работанных новых (файловой и многомерно-матричной) моделей данных, методов доказательства соответствия модели данных и модели вычислений, методов синтеза и оптимизации процессов массовой обработки данных, становится возможным построение программно-аппаратных комплексов для решения важных задач параллельной реализации массовой обработки данных.

Практическая значимость результатов, полученных в данной работе состоит в следующем:

1. Предложенные подходы, методы и алгоритмы могут быть использованы для проектирования и разработки программно-аппаратных комплексов МОД на базе широкого спектра многоядерных и многопроцессорных систем с архитектурами SMP, MPP, NUMA, таких как стоечные серверы и локальные грид-системы.

2. Предложенные алгебраические модели и методы могут быть использованы для разработки технологии коллективного проектирования больших многопроцессорных программно-аппаратных комплексов и для создания сложных программных систем различного назначения.

3. На разработанные модели и методы получены патенты:

3.1. Патент на полезную модель RUS 82355 12.08.2008/ Система представления данных в базе данных Сергеев В.П., Гайдаенко Т.И., Левин Н.А., Мунерман В.И., Оздемир С.М., Провоторова А.О., Ширай А.Е. (приложение 1).

3.2. Патент № 2755568 Российская Федерация, МПК G06F 16/2455. Способ параллельного выполнения операции JOIN при обработке больших структурированных высокоактивных данных: №2020124733: заявл. 26.07.2020: опубл. 17.09.2021 / Мунерман В. И.,Синявский Ю. В., Чукляев И. Л., Чукляев Е. И. – 10 с. На этот патент подана «Международная заявка PCT/RU2021/000480 от 02.11.2021 г.» (приложение 2).

4. На разработанное на основе алгебраических моделей и методов программное обеспечение получены авторские свидетельства (приложение 3):

4.1. Свидетельство о регистрации программы для ЭВМ 2020613833 "Программа для обработки распределенных больших объемов данных для стоечных

серверов и дата-центров".

4.2. Свидетельство о регистрации программы для ЭВМ 2020614270 "Программа для обработки распределенных больших объемов данных в вычислительных сетях рабочих станций".

Достоверность результатов

Достоверность полученных в диссертационной работе результатов подтверждается проведенными натурными испытаниями программного обеспечения, реализующего разработанные методы и алгоритмы.

Апробация работы

Основные положения диссертационной работы, разработанные модели, методы, алгоритмы и результаты вычислительных экспериментов многократно докладывались на международных и всероссийских научных конференциях, в том числе:

– Международная научная конференция Системы компьютерной математики и их приложения, г. Смоленск, Смоленский государственный университет, 2008-2024 гг.

– Международная научно-практическая конференция «Современные информационные технологии и ИТ-образование», г. Москва, Московский государственный университет, 2014-2017 гг.

– Международная научная конференция «Конвергентные когнитивно-информационные технологии» в рамках международного конгресса «СОВРЕМЕННЫЕ ПРОБЛЕМЫ КОМПЬЮТЕРНЫХ И НФОРМАЦИОННЫХ НАУК», г. Москва, Московский государственный университет, 2018-2023 гг.

– IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), г. Зеленоград, Национальный исследовательский университет «МИЭТ», 2019-2021 гг.

– III научно-практическая конференция с международным участием «Актуальные проблемы информатизации в цифровой экономике и научных исследованиях – 2022» г. Зеленоград, Национальный исследовательский университет «МИЭТ».

– XVII International conference «Data Analytics and Management in Data Intensive Domains», DAMDID/RCDL'2015, г. Обнинск.

Личный вклад соискателя

В диссертационной работе приведены научные положения и практические результаты, полученные лично автором. В том числе:

– соискателем впервые предложена возможность использования алгебры многомерных матриц для моделирования процессов массовой обработки данных;

– введены понятия коэффициента активности данных и высокой активности данных;

– приведено доказательство возможности параллельной реализации умножения многомерных матриц как распределенной совокупности их сечений по скоттовым индексам;

– проведено исследование предложенного соискателем алгоритма симметричного горизонтального распределения данных для параллельной реализации операций типа Inner Join.

Из совместных публикаций в диссертацию включен лишь тот материал, который непосредственно принадлежит соискателю, заимствованный материал обозначен в работе ссылками.

Публикации автора по теме диссертации

По результатам диссертационной работы автором опубликовано 73 работы, в том числе 33 работы в журналах, входящих в список научных журналов ВАК Минобрнауки России. И них в течении последних пяти лет 2 в изданиях, входящих в список К1, 31 в изданиях, входящих в список К2; 9 работ в изданиях, индексируемых в международных наукометрических базах Scopus и Web of Science, 1 монография и 2 патента Российской Федерации.

Структура и объем диссертации

Диссертация состоит из введения, шести глав, заключения и библиографии. Объем диссертации составляет 286 страниц, объем библиографии – 244 наименования.

Глава 1. МАССОВАЯ ОБРАБОТКА ДАННЫХ: ОПРЕДЕЛЕНИЕ, МЕТОДЫ ОПИСАНИЯ, АНАЛИЗ ПРОБЛЕМ

1.1. Понятие массовой обработки данных

Традиционно массовую обработку данных связывают с параллельными вычислениями и чаще всего определяют следующим образом: массовая параллельная обработка – способ параллельной обработки больших объемов данных большим числом процессоров. В настоящее время массовую обработку данных связывают с направлением, получившим название Big data. Big data (большие данные) – общий термин, который обозначает вновь создающиеся структурированные, неструктурированные и полуструктурированные данные сверхбольших и постоянно возрастающих объемов; загрузка их в обычную (например, реляционную) базу данных и последующая обработка требуют слишком больших затрат ресурсов вычислительных комплексов. В работе рассматривается один из классов массовой обработки – обработка структурированных данных.

Исторически, обработка структурированных данных – один из самых ранних классов обработки. Первые математические (алгебраические) модели для него появились еще в начале 60-х годов XX века [61, 62] и, в конечном счете, привели к современным реляционным и объектным моделям данных [63]. Технологические решения также развивались, в основном, применительно к обработке структурированных данных. К числу таких решений можно отнести формализацию методов доступа к данным. Разделение их на последовательный, индексный и индексно-последовательный позволило существенно повысить производительность вычислительных комплексов, так как метод доступа определялся характером решаемой задачи. Это позволяло выбирать наиболее эффективный алгоритм обработки данных для каждой операции из последовательности операций, приводящих к решению прикладной задачи. Далее в работе речь будет идти только о массовой обработке структурированных данных, поэтому под термином "массовая обработка данных" (МОД) будет пониматься только обработка структурированных данных.

Традиционно МОД широко используется для решения многих задач в различных предметных областях в тех случаях, когда в вычисления включается значительная часть данных. К числу таких задач относятся, например:

– оперативная статистическая обработка экспериментальных данных, таких, как виброзащитные характеристики, оперативно получаемые в ходе летных испытаний [64];

– в банковской сфере [65. 66], в частности, задача "Операционный день банка" требует ежедневной обработки от 40% до 90% всей базы данных банка;

– ежедневные задачи учета и планирования производства в современных системах управления (стандарты ERP [67, 68]), при решении которых процент обрабатываемых данных всегда близок к 100;

– задачи статистического анализа и синтеза подсистем послепродажного обслуживания в системах интегрированной логистической поддержки наукоемкой продукции [69].

Таким образом, особенность этих классов задач заключается в том, что:

1. при их решении в обработку включаются практически все данные, характеризующие объекты этих задач;

2. объемы обрабатываемых данных очень велики, то есть можно утверждать, что они (эти классы) относятся к области исследований, связанной с обработкой данных больших объемов (big data).

В работе рассматривается такая разновидность МОД, которая позволяет учесть эти особенности и, основываясь на свойствах и структурах данных, присущих рассмотренным классам задач, обеспечивает эффективную реализацию вычислительных процессов решения этих задач.

Далее предполагается, что используемые и обрабатываемые в задачах МОД данные хранятся в базах данных (БД) и обрабатываются системами управления базами данных (СУБД). Под СУБД понимается, в соответствии с международными стандартами [70, 71], программная система, предназначенная для создания и хранения базы данных на основе некоторой модели данных, обеспечения логической и физической целостности содержащихся в ней дан-

ных, надежного и эффективного использования ресурсов (данных, пространства памяти и вычислительных ресурсов). В работе рассматриваются именно методы эффективной реализации МОД с использованием современных аппаратных и программных средств. Повышение эффективности достигается за счет предложения и использования специальных моделей данных.

Обычно СУБД опирается на файловую систему, присущую конкретному вычислительному комплексу или операционной системе [72]. Файловая система обеспечивает функции управления данными, хранимыми в файлах во внешней памяти. Эти данные организуются с использованием различных методов доступа, которые трактуются как совокупность соглашений о способах размещения данных некоторого типа в пространстве памяти, поиска требуемых экземпляров и выполнения над ними операций навигации, выборки обновления и удаления [73]. Однако при работе с СУБД методы управления файлами, определяющие способы их организации и доступа к отдельным записям, уходят на второй план и становятся невидимыми для программистов, разрабатывающих запросы к базе данных. Это приводит к тому, что способы повышения эффективности обработки данных (оптимизация запросов) перестают быть инструментами прикладного программиста и становятся прерогативой программистов, разрабатывавших СУБД. Это подтверждается такими стандартными определениями запроса как:

– "вопроса, затрагивающего те или иные аспекты информации, хранящейся в базе данных, и модификации средствами языка запроса или языка манипулирования данными" [70];

– "сообщения конечного пользователя или приложения, направляемого СУБД и активизирующего в системе баз данных действия, обеспечивающие выборку, вставку, удаление или обновление указанных в нем данных" [71].

Частичные решения, дающие прикладному программисту некоторую свободу действий, появились благодаря объектным моделям данных. В большинстве случаев эти решения ограничиваются представляемой ему возможностью создания собственных абстрактных типов данных (классов, объектов). Такой

подход позволяет разрабатывать эффективные процедуры, реализующие операции над сложными нестандартными типами данных. Но главная проблема МОД – построение эффективного процесса обработки данных при помощи некоторого набора типовых операций [73, 74], – в современном понимании объектно-ориентированного подхода к базам данных не учитывается. Это не позволяет прикладным программистам активно влиять на стратегию и тактику оптимизации обработки данных.

Поэтому цели работы, направленные на повышение эффективности МОД, определяются следующим образом:

1. разработка моделей данных, обеспечивающих наибольшее соответствие существующим моделям данных и архитектурам вычислительных комплексов;

2. разработка на основе этих моделей способов организации данных во внешней памяти и алгоритмов реализации операций, в наибольшей степени соответствующих структурам данных.

На основе сказанного можно сформулировать основные предположения и определить (пока нестрого) понятия, в наибольшей степени соответствующие целям работы:

1. данные хранятся в базе данных (БД) и управляются системой управления базами данных (СУБД);

2. модель данных и способ их организации, присущие конкретной СУБД, не влияют на характер выборки в процессе выполнения операций;

3. данные извлекаются из базы в виде поименованных упорядоченных последовательностей однотипных агрегатов, содержащих сведения об однородных объектах;

4. в дальнейшем для этих последовательностей будет использоваться (в соответствии с традицией) название ***файлы***, а для агрегатов – ***записи***;

5. предполагается, что время выборки данных из базы в файл минимально (на практике оно определяется свойствами СУБД).

Вовлеченность большинства записей в обработку можно выразить количественно при помощи величины, называемой коэффициентом активности

файла [73. 75]. Пусть L – общее число записей в файле, L_e – число записей того же файла, подвергающихся обработке в процессе выполнения операции над этим файлом.

Отношение $K_a = \frac{L_e}{L}$ называется ***коэффициентом активности файла***. Коэффициент активности определяет метод доступа к файлу. Если он высокий – близок к единице, то более эффективным будет последовательный доступ к файлу, в противном случае, когда он близок к нулю, целесообразно использовать произвольный доступ.

Таким образом, основное свойство МОД состоит в том, что коэффициенты активности всех обрабатываемых файлов близки к единице.

Реализация МОД осуществляется посредством обработки файлов специальным набором операций. Причем этот набор операций не меняется на протяжении длительного периода времени от конца пятидесятых годов XX века до настоящего времени. Одним из наиболее эффективных подходов к формализации был подход, основанный на создании алгебры файлов [74]. При таком подходе файл рассматривается как самостоятельный объект, обладающий набором присущих ему специфических свойств, существенно используемых при формализации операций и разработке алгоритмов, реализующих эти операции. В [73, 74] формализуется понятие ключа, как неотъемлемого свойства файла, а файлы и операции классифицируются в соответствии со свойствами заданных на них ключей. Такой подход имеет ряд преимуществ перед появившемся несколько позже, но завоевавшем исключительную популярность реляционным подходом. Одно из этих преимуществ состоит в том, что файл определяется и рассматривается в контексте операций не как множество, подобно тому, как это делается в реляционном подходе при определении отношения, а как производный от множества объект, что позволяет избежать использования громоздкого аппарата нормализации. Второе преимущество заключается в том, что формализация файлов и операций над ними позволяет создавать процедурные модели данных, в которых определения операций одновременно задают способы представления

данных и алгоритмы их обработки. Вместе с тем оба подхода не только не противоречат друг другу, но и взаимно дополняют друг друга, что особенно важно в настоящее время, когда объектно-ориентированный подход к обработке данных, наиболее естественный с математической точки зрения, стал преобладающим. Поэтому, а также в силу популярности реляционного подхода и производных от него, в дальнейшем при описании представления данных и операций, над ними будут приводиться их аналоги в реляционной (SQL) модели.

Далее приводится неформальное описание операций над файлами в МОД, строгое определение которых будет дано в последующих главах. Эти описания впервые были приведены в книге [62].

1. Сортировка. Упорядочивает файл в соответствии с некоторым отношением порядка (как правило, лексикографического), заданного на множестве записей файла. Поля записей файла, по значениям которых упорядочивается файл, называются *ключами сортировки*, *ключевыми полями* или просто *ключами.* В реляционной модели данных нет явной операции сортировки, но в языке SQL есть возможность упорядочить результат запроса.

2. Выборка. Выбирает из файла записи, соответствующие заданному критерию. В реляционной модели ей соответствует операция Select.

3. Сжатие. Операция квантификации. Заменяет несколько записей, удовлетворяющих заданному критерию, одной записью. При этом часть элементов (полей) записей могут подвергаться групповым операциям, смысл и алгоритмы которых определяются контекстом операции сжатия в предметной области, в которой решается задача. В реляционной модели этой операции соответствует операция Project, которая реализуется в языке SQL возможностью выборки не всех полей отношения и применения операции GROUP BY.

4. Слияние строго упорядоченных файлов. На самом деле это не одна операция, а класс операций, соответствующий классу теоретико-множественных операций в реляционной модели. Отличие состоит в том, что оба файла рассматриваются как упорядоченные множества. Таким образом, слияние строго упорядоченных файлов позволяет реализовать операции подобные объедине-

нию, пересечению, разности и симметрической разности множеств. Однако строгое определении этой операции дает возможность прикладному программисту конструировать и другие варианты получения результирующего файла, что в большей степени соответствует концепциям объектно-ориентированного подхода.

5. Слияние нестрого упорядоченных файлов. Для этой операции по крайней мере один из участвующих в ней файлов рассматривается как мультимножество (множество, допускающее включение одного и того же элемента по нескольку раз). Ее алгоритм состоит из следующих шагов:

Шаг 1. В обоих исходных файлах выбираются группы записей, одинаковых по заданному критерию.

Шаг 2. Выполняется декартово произведение выбранных групп. Каждая полученная пара записей заменяется единственной записью файла результата.

Шаг 3. Результаты декартовых произведений объединяются в один файл.

В реляционной модели этой операции соответствует операция JOIN, та ее разновидность, которая называется естественным соединением.

Операцию слияния строго упорядоченных файлов и операцию слияния нестрого упорядоченных файлов, также как и соответствующие им реляционные операции, можно рассматривать как аддитивную и мультипликативную соответственно.

Далее приводятся примеры задач, для решения которых целесообразно применение МОД.

Пример 1.1. Типичный пример задачи, решение которой требует применения МОД, относится к системам интегрированной логистической поддержки наукоемкой продукции (CALS). Это задача снабжения и комплектации сервисных центров для проведения технического обслуживания (ТО) и ремонта [77].

Исходные данные хранятся в файлах:

- **План**(Изделие, Вид_ТО, Дата_нач, Дата_кон);
- **Сроки**(Заказчик, Изделие, Вид_ТО, Дата, Количество);
- **Комплект**(Изделие, Вид_ТО, Ремкомплект).

Файлы **План** и **Сроки** лексикографически нестрого упорядочены по значениям пары полей Изделие и Вид_ТО их записей. Это означает, что каждый файл содержит не менее одной записи, в каждой из которых поля Изделие и Вид_ТО содержат одинаковые значения. Операция слияния этих нестрого упорядоченных файлов, порождает файл **Факт**(Заказчик, Изделие, Вид_ТО, Дата, Количество). Записи файла **Факт** формируются в процессе вычисления декартовых произведений в том случае, когда поле Дата в записи файла **Сроки** находится в интервале, заданном полями записи Дата_нач и Дата_кон файла **План**.

Файлы **Факт** и **Комплект** также нестрого упорядочены по значениям полей Изделие и Вид_ТО. Поэтому файл-результат **Комплект_ТО**(Заказчик, Изделие, Вид_ТО, Ремкомплект, Дата, Количество) также может быть получен как результат операции слияния этих нестрого упорядоченных файлов.

Записи файла **Комплект_ТО** всегда порождаются из записей файлов **Факт** и **Комплект** во время вычисления декартовых произведений. При таком подходе файл **Комплект_ТО** остается нестрого упорядоченным и поэтому должен быть подвергнут операции свертки. На практике, свертка выполняется в ходе операции слияния нестрого упорядоченных файлов в ходе формирования построения декартовых произведений.

Для решения задачи можно также использовать реляционную модель данных. В этом случае исходные документы представляются в виде трех отношений: **План**, **Сроки** и **Комплект**, аналогичных соответствующим файлам. Задача решается с помощью двух SQL-запросов.

В результате первого запроса:

```
SELECT
    Сроки.Заказчик, Сроки.Изделие, Сроки.Вид_ТО, Сроки.Дата,
    Сроки.Количество
FROM Сроки INNER JOIN План ON
    (Сроки.Вид_ТО = План.Вид_ТО) And
    (Сроки.Изделие = План.Изделие)
WHERE
```

((([Сроки]![Дата]) ***Between*** [План]![Дата_нач] ***And*** План]![Дата_кон]));

формируется отношение **Факт** (Заказчик, Изделие, Вид_ТО, Дата, Количество).

Второй запрос:

SELECT

Факт.Заказчик, Факт.Изделие, Факт.Вид_ТО, Комплект.Ремкомплект,

Min(Факт.Дата) **AS** [Дата], ***Sum***(Факт.Количество) **AS** [Количество]

FROM Факт **INNER JOIN** Комплект **ON**

(Факт.Вид_ТО = Комплект.Вид_ТО) ***And***

(Факт.Изделие = Комплект.Изделие)

GROUP BY

Факт.Заказчик, Факт.Изделие, Факт.Вид_ТО, Комплект.Ремкомплект;

приводит к отношению-результату **Комплект_ТО**(Заказчик, Изделие, Вид_ТО, Ремкомплект, Дата, Количество).

В примере 1.1 приведены словесное описание и алгебраическая (на языке SQL) форма процесса (алгоритма) вычисления файла (отношения) **Комплект_ТО**. Очевидно, что при больших объемах данных, содержащихся в исходных файлах (отношениях), время последовательной реализации этого процесса будет настолько большим, что по завершении вычислений полученные данные могут оказаться неактуальными. Поэтому естественный выход из ситуации заключается в применении методов, основанных на использовании параллельных и распределенных систем баз данных.

Параллельная система баз данных поддерживает очень большие базы данных и использует свойства параллелизма архитектуры мультипроцессорных вычислительных комплексов, в средах которых она функционирует [55, 77, 78].

Распределенная система баз данных функционирует в сетевой среде, и данные в ней физически распределены между несколькими узлами. На каждом узле данные управляются собственной СУБД, которая работает независимо от СУБД, используемых на других узлах [79-81].

Одна из важнейших проблем этих видов систем баз данных – оптимизация запроса. Решение этой проблемы возможно на интуитивном уровне, опи-

рающемся на опыт и искусство прикладного программиста. Далее рассмотрены два примера, иллюстрирующие возможность реализации, рассмотренной в примере 1.1 задачи МОД на различных архитектурах параллельных вычислительных комплексов.

Пример 1.2. Для решения задачи можно использовать конвейерную архитектуру вычислительного комплекса, относящуюся к классу MISD (множественный поток команд, одиночный поток данных) [82, 83], где несколько функциональных модулей (два или более) выполняют различные операции над одними данными. Архитектура такого комплекса приведена на рисунке 1.1. Вычислитель 1 выполняет операцию слияния нестрого упорядоченных файлов **План** и **Сроки**. После того, как сформирован очередной фрагмент файла **Факт** (результат вычисления декартова произведения), он передается Вычислителю 2. Вычислитель 2 вычисляет декартово произведение этого фрагмента и соответствующей ему группы записей файла **Комплект,** а затем сохраняет его в файле-результате **Комплект_ТО**.

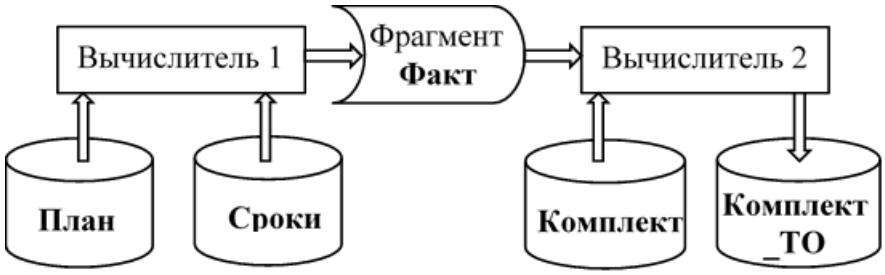


Рис. 1.1. Конвейерный вычислительный комплекс для решения задачи снабжения и комплектации сервисных центров

Эта архитектура может быть расширена (масштабирована) для решения задачи слияния произвольного числа N нестрого упорядоченных исходных файлов. В этом случае потребуется N–1 вычислитель. При этом существенное значение будет иметь последовательность выполнения операций. В последующих главах будет доказано, что получение оптимальной последовательности возможно методом динамического программирования, подобно тому, как это сделано в [84]. Кроме того, поскольку существуют различные способы параллельной реализации операции декартова произведения множеств, то вычисли-

тели, составляющие конвейер, также могут быть параллельными вычислительными комплексами.

Пример 1.3. Решение рассмотренной задачи может быть получено на вычислительном комплексе с архитектурой SIMD (одиночный поток команд, множественный поток данных) [82, 83], представленной на рисунке 1.2. В этой архитектуре несколько (более двух) вычислителей, имеющих свою собственную (в некоторых случаях общую) оперативную и внешнюю память, связаны с вычислительной машиной, осуществляющей общее управление (хост-машиной). В современных комплексах возможны связи и между вычислителями. Тройки групп файлов **План**, **Сроки** и **Комплект**, которые содержат одинаковые значения полей Изделие и Вид_ТО, собираются на одном внешнем запоминающем устройстве, связанном с одним из *N* вычислителей.

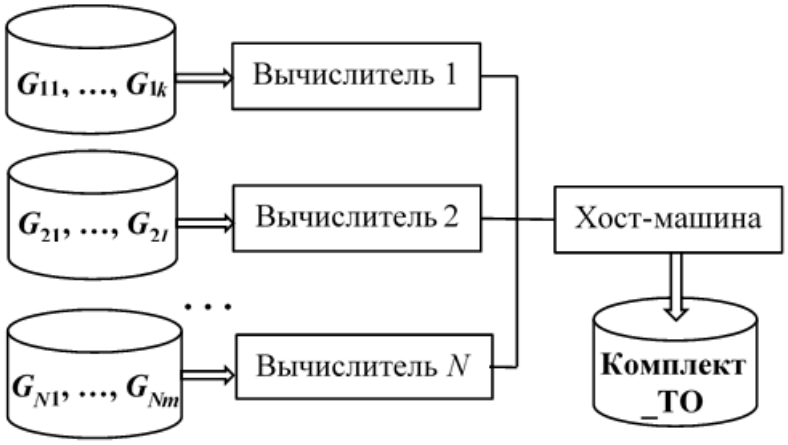


Рис. 1.2. SIMD-вычислительный комплекс для решения задачи снабжения и комплектации сервисных центров

Если число троек равно *N*, то на каждом запоминающем устройстве будет единственная тройка. В общем случае, число $M>N$ таких троек на каждом запоминающем устройстве может быть больше единицы ($k + l + \ldots + m = M$).

Каждый вычислитель выполняет одинаковую для всех вычислителей операцию построения декартовых произведений групп записей троек, расположенных в связанной с ним внешней памяти. Полученные в результате записи файла **Комплект_ТО** передаются на хост-машину, где собираются в результирующий файл.

Поскольку возможна ситуация $M>N$, то одна из важных задач состоит в том, чтобы вычислители были загружены равномерно. Для этого требуется распределить тройки по внешним запоминающим устройствам таким образом, при котором объемы занимаемой ими памяти на всех устройствах были бы примерно одинаковыми. Алгоритмы распределения данных рассмотрены в последующих главах.

Рассмотренные примеры показывают возможность параллельной реализации задач МОД для структурированных и высоко активных данных. Методы параллельной реализации задачи МОД, рассмотренные в примерах, могут быть реализованы не только в рамках файловых систем, но и средствами любых СУБД. При этом модель данных, на которой основана СУБД (реляционная, объектная, SQL, NoSQL), не имеет принципиального значения. Однако эти модели не учитывают основные факторы, определяющие эффективность МОД: структурированность и активность данных. Они разрабатывались как универсальные модели, пригодные для решения любых классов задач обработки данных. Но проблемы распараллеливания данных и оптимизации запросов неотделимы от знания структуры данных и оценки вычислительной сложности алгоритмов, которая определяется величиной входного потока [85]. Поэтому невозможно разработать общие методы распараллеливания и оптимизации обработки, которые бы позволяли одинаково хорошо оптимизировать обработку данных для всех возможных классов задач.

Для того, чтобы иметь возможность построения технологий распараллеливания обработки данных и решать проблемы оптимизации запросов для рассматриваемого класса МОД, необходимо построение моделей, учитывающих эти факторы: структурированность и активность данных. Поэтому дальнейшие главы посвящены построению таких моделей. В главе 2 построена и рассмотрена теоретико-множественная модель данных, а в главе 3 многомерно-матричная модель.

1.2. Связь моделей МОД с архитектурами программно-аппаратных комплексов

Для того чтобы понять, какая архитектура вычислительного комплекса наиболее подходит для реализации задач МОД, необходимо определить методы хранения и доступа к данным. В этом параграфе дается описание этих методов и определяется критерий вычислительной сложности операций, предложенных в 1.1. Также рассматриваются архитектуры вычислительных комплексов, на которых эти задачи могут быть эффективно решены и конкретные технические решения, реализующие эти архитектуры.

1.2.1. Логически последовательный метод доступа

Характерная особенность рассматриваемой в работе разновидности МОД заключается в том, что данные, будучи высоко активными в контексте операций их обработки, в незначительной мере подвержены изменениям. Как правило, изменения накапливаются в специальном файле – корректуре, и перед решением основных задач производится необходимая корректировка файлов. Операция корректировки файла реализуется при помощи слияния строго упорядоченных файлов. Эта особенность позволяет рассматривать все файлы, участвующие в процессе обработки данных, как последовательные и упорядоченные. Упорядоченность обеспечивает ускорение выполнения всех определенных в параграфе 1.1 операций. В самом медленном случае, когда при реализации не используется параллелизм, упорядоченность файлов позволяет выполнять все операции за один проход (просмотр) исходных и выходного файлов. При использовании параллельных вычислительных комплексов упорядоченность обеспечивает лучшее распределение файлов и их фрагментов по ресурсам комплекса и эффективное использование этих ресурсов. Повышение эффективности обработки данных при такой организации данных происходит по следующим причинам:

– файл занимает меньше места на внешнем запоминающем устройстве, поскольку нет необходимости использовать списковую структуру и хранить удаленные записи (мусор).

– уменьшается время чтения/записи, поскольку файл в идеальном случае занимает непрерывное пространство, но даже при необходимости фрагментации, если обслуживание внешней памяти организовано правильно, число фрагментов невелико. Поскольку внешняя память относится к типу блочных устройств ввода-вывода, за одно обращение к ней читается/записывается блок, содержащий большое число записей.

– упрощается обслуживание данных, так как становятся ненужными операции сборки мусора и переформирования файлов.

При таком подходе прикладная программа читает/пишет файл последовательно по одной записи, то есть для МОД характерны последовательная организация файла и последовательный доступ к нему. Однако при работе с СУБД невозможно точно знать, какие способы организации данных и методы доступа в них используются. Поэтому в дальнейшем речь будет идти о *логически последовательной* организации файла и *логически последовательном* методе доступа [85, 86]. Это позволяет учитывать тот факт, что способы организации данных и доступа к ним в различных СУБД могут существенно различаться. Логически последовательный метод доступа обеспечивает реализацию всех операций над файлами, но в случае операции слияния нестрого упорядоченных файлов он может приводить к нерациональному использованию ресурсов вычислительного комплекса. При последовательной реализации этой операции требуется либо многократное повторное считывание одного из файлов, либо большой объем дополнительной оперативной памяти. При параллельной организации возникают проблемы, связанные с разделением частей файлов между вычислителями.

Выход из положения возможен благодаря применению индексно-последовательного метода доступа в соответствии со следующими правилами организации основного и индексного файлов:

1. основной файл данных упорядочен по значениям ключа $K_1, \ldots, K_M$;

2. индексный файл упорядочен аналогично;

3. запись индексного файла содержит три поля: поле Ключ, которое может быть составным полем, поле Индекс, содержащее номер первой записи с дан-

ным значением ключа, и поле Счетчик, содержащее количество записей с данным значением ключа.

Взаимодействие индексного и основного файлов при индексно-последовательном методе доступа показано на рисунке 1.3.

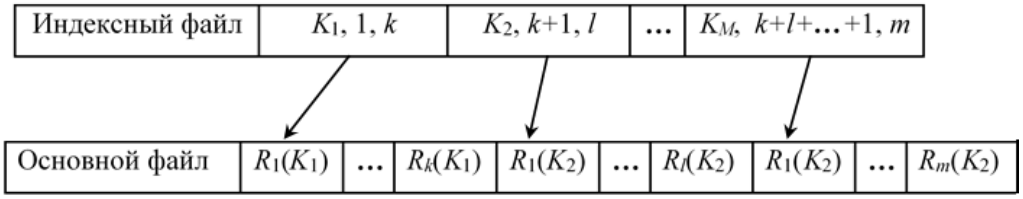


Рис. 1.3. Индексно-последовательная организация файла

Для оценки эффективности процесса предлагается мера, связанная с просмотром исходных файлов и выводом выходного файла. Для некоторых устройств, из которых конструируется внешняя память, время чтения и время записи данных одинаковы, для некоторых, таких как SSD, различны. Но, в любом случае, это время зависит от числа записей в файле. Предлагаемая мера также не зависит от способов реализации операций обработки файлов, но непосредственно определяет порядок затрат машинного времени и называется *длиной просмотра*. Временем, которое тратится процессором на обработку прочитанных данных, можно пренебречь, так как в большинстве современных вычислительных комплексов операции обмена выполняются параллельно с вычислениями.

Длина просмотра позволяет оценить сложность операций логически последовательной обработки данных:

– операция сортировки (балансной) имеет порядок, определяемый величиной $\frac{L\log_2 L}{b}$ [84], где L – длина файла, b – размер считываемой в оперативную память порции записей сортируемого файла; L и b измеряются в одних и тех же единицах, например, в байтах или записях;

– унарные операции (выборки и сжатия) – величиной L_s+L_r, равной сумме длин просмотров исходного файла и файла результата;

– бинарные операции (слияний строго и нестрого упорядоченных файлов) – величиной $L_{s_1} + L_{s_2} + L_r$, равной сумме длин просмотров обоих исходных файлов и файла результата.

Сложность операции может быть увеличена на сложность сортировки одного или обоих исходных файлов, если сортировка необходима для эффективного выполнения операции. При параллельной реализации операций МОД вычислительная сложность операций уменьшается. Так при выполнении операции слияния нестрого упорядоченных файлов время ее реализации будет определяться длиной просмотра наибольшего фрагмента исходных и выходного файлов, то есть временем работы одного из вычислителей комплекса. Подробно определение вычислительной сложности операций будет рассмотрено в последующих главах.

1.2.2. Архитектуры вычислительных комплексов для реализации логически последовательного метода доступа

В соответствии с классификацией Флинна [82, 83], отражающей взаимодействие программ и данных, параллельные вычислительные комплексы имеют следующие архитектуры.

SIMD компьютер имеет N идентичных процессоров, N потоков данных и один поток команд. Каждый процессор обладает собственной локальной памятью. Процессоры интерпретируют адреса данных либо как локальные адреса собственной памяти, либо как глобальные адреса, возможно, модифицированные добавлением локального базового адреса. Процессоры получают команды от одного центрального контроллера команд и работают синхронно, то есть на каждом шаге все процессоры выполняют одну и ту же команду над данными из собственной локальной памяти. Такая архитектура с распределенной памятью часто упоминается как архитектура с параллелизмом данных (data-parallel), так как параллельность достигается при наличии одиночного потока команд, действующего одновременно на несколько частей данных.

MISD компьютер характеризуется множественным потоком команд и одинарным потоком данных. Эта архитектура называется также конвейером обработки данных. MISD компьютер представляет собой цепочку из N последовательно соединенных процессоров (не обязательно идентичных), которые управляются параллельным потоком команд. На вход конвейера из памяти подается одинарный поток данных, которые проходят последовательно через все процессоры. Каждый процессор производит обработку данных под управлением своего потока команд и передает результаты следующему по цепочке процессору, который использует их как входные данные.

Большинство современных вычислительных комплексов можно отнести к классу MIMD, объединяющему свойства двух предыдущих классов.

MIMD компьютер имеет N процессоров, независимо исполняющих N потоков команд и обрабатывающих N потоков данных. Каждый процессор функционирует под управлением собственного потока команд, то есть MIMD компьютер может параллельно выполнять совершенно разные программы.

Из примеров, рассмотренных в 1.1, следует, что все эти классы архитектур могут быть использованы для реализации МОД. Выбор конкретной архитектуры как способа взаимодействия программы и данных определяется прикладным программистом в процессе проектирования и основывается на знании структуры данных в конкретной задаче.

В классификации, определяющей взаимодействие ресурсов вычислительного комплекса [87, 88] для реализации МОД наибольший интерес представляют следующие архитектуры.

Симметричное мультипроцессирование (SMP) – это архитектура многопроцессорных компьютеров, в которой два или более одинаковых процессоров подключаются к общей оперативной и внешней памяти.

В настоящее время в результате применения многоядерных (multicore) и гиперпоточных (hyper threading) процессоров, практически все вычислительные комплексы, начиная с простейших мобильных устройств и заканчивая суперкомпьютерами, либо сами по себе симметричные мультипроцессорные вычис-

лительные комплексы, либо содержат элементы, реализованные в этой архитектуре. Типичный пример такого подхода – рассмотренная далее машина баз данных IBM Netezza и Oracle Exadata, в которых совмещены SMP и MPP архитектуры [89-92].

Массивно-параллельные комплексы (MPP) – эта архитектура объединяет как неоднородные, так и однородные вычислительные комплексы. В простейшем случае – это комплекс с одним или несколькими многоядерными процессорами, единственное требование к которой состоит в том, чтобы каждый процессор (ядро) был ассоциирован с собственным устройством массовой памяти (в простейшем случае – дисковым накопителем). Наиболее популярные представители этой архитектуры – вычислительные комплексы, построенные по кластерной технологии [93] и грид-технологии [94, 95].

Кластер – это группа компьютеров, объединённых высокоскоростными каналами связи и представляющая с точки зрения пользователя единый аппаратный ресурс.

Грид – это согласованная, открытая и стандартизованная среда, которая обеспечивает гибкое, безопасное, скоординированное разделение (общий доступ) ресурсов в рамках виртуальной организации.

Основное отличие грид-систем от кластеров состоит в отсутствии требования компактного размещения узлов комплекса, то есть они могут быть настолько удалены друг от друга, что для их взаимодействия используются глобальные сети, например, Интернет. Это может отрицательно повлиять на время решения задач.

Далее рассматриваются два вычислительных комплекса, которые можно отнести к классу машин баз данных, и которые могут быть эффективно использованы для реализации МОД. Это IBM Netezza и Oracle Exadata.

По мнению разработчиков, архитектура IBM Netezza сочетает в себе лучшие свойства симметричного мультипроцессирования (SMP) и массовой параллельной обработки (MPP). Высокопроизводительный сервер баз данных Netezza Performance Server (NPS) имеет двухуровневую архитектуру (рисунок

1.4), называемую архитектурой асимметричной массовой обработки данных (Asymmetric Massively ParallelProcessing™ (AMPP™) architecture).

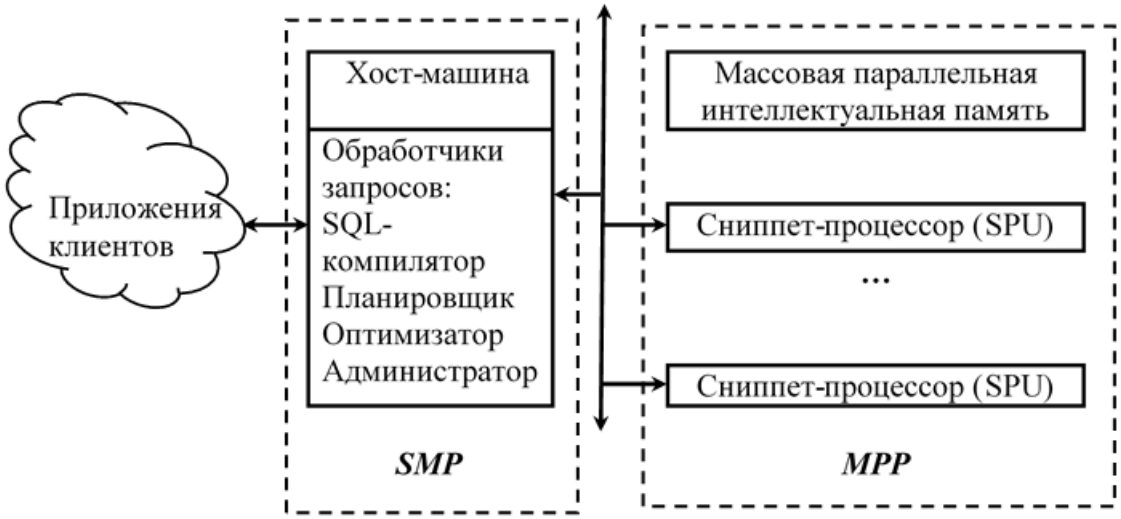


Рис. 1.4. Архитектура машины баз данных IBM Netezza

Первый уровень – это симметричная многопроцессорная хост-машина, связанная по сети с клиентами, на которых выполняются различные прикладные программы, генерирующие запросы к базам данных. Программное обеспечение этого уровня выполняет компиляцию SQL-запросов, а затем планирование, оптимизацию и администрирование их выполнения. Запрос делится на последовательность подзадач, или фрагменты, которые могут выполняться параллельно. Эти фрагменты передаются на второй уровень для исполнения. По окончании выполнения всех фрагментов хост-машина формирует окончательный результат, который передается по сети клиентской прикладной программе, выдавшей запрос, таким же образом как это делают традиционные серверы баз данных.

Рис. 1.5. Архитектура сниппет-процессора

Второй уровень связан с первым высокоскоростной сетью и состоит из большого числа (от десятков до тысяч) специализированных процессоров, ко-

торые называются Snippet Processing Unit (SPU), или сниппет-процессоров, архитектура которых показана на рисунке 1.5.

Каждый SPU представляет собой интеллектуальный узел, реализующий хранение данных и обработку запросов. Он состоит из центрального процессора, оперативной памяти, сетевого контроллера, обеспечивающего связь с хост-машиной по высокоскоростной сети, интеллектуального контроллера диска, реализованного на основе программируемой логической интегральной схемы (ПЛИС/FPGA) и дисковой памяти. Интеллектуальный контроллер диска позволяет реализовать технологию ускорения потоков данных (Accelerated Streaming Technology (FAST)). Ускорение достигается за счет использования специального набора инструкций, ориентированного на частичное выполнение операций языка SQL на аппаратном уровне. Имеется возможность гибко добавлять новые инструкции.

Данные распределяются по SPU, с помощью хэш-функции. Первичная обработка запросов осуществляется на уровне SPU, причем каждый из них выполняет обработку своей части базы данных. В основном выполняются операции, которые легко поддаются параллельной обработке, например, операции над записями, такие как синтаксический анализ, фильтрация, проектирование, блокировка и им подобные. Это позволяет значительно уменьшить количество данных, перемещаемых внутри комплекса. Операции над промежуточными результатами, такие как сортировка, объединение и агрегирование, также выполняются в первую очередь на SPU, и только потом доводятся до окончательного завершения на хост-машине. Термин сниппет-процессор имеет двоякий смысл. Во-первых, эти процессоры выполняют на аппаратном уровне процедуры, реализующие части SQL-запросов, во-вторых, каждый из них работает со своим фрагментом базы данных.

С точки зрения прикладного программиста, Oracle Exadata представляет собой единое устройство, функционирующее как машина баз данных. Архитектурно – это (рисунок 1.6) программно-аппаратный вычислительный комплекс, в котором объединены серверы хранения данных (Exadata Storage Servers), на ко-

торых могут располагаться как локальные, так и распределенные базы данных, и программное обеспечение, реализующее за счет кластеризации повышение доступности к данным в СУБД Oracle (Oracle Real Application Clusters (RAC)). Для связи между узлами этого комплекса использована Infiniband – высокоскоростная коммутируемая последовательная шина, применяющейся в данном случае для внутренних соединений. По мнению разработчиков, прикладной программист может рассматривать Oracle Exadata как специфическое запоминающее устройство для хранения данных.

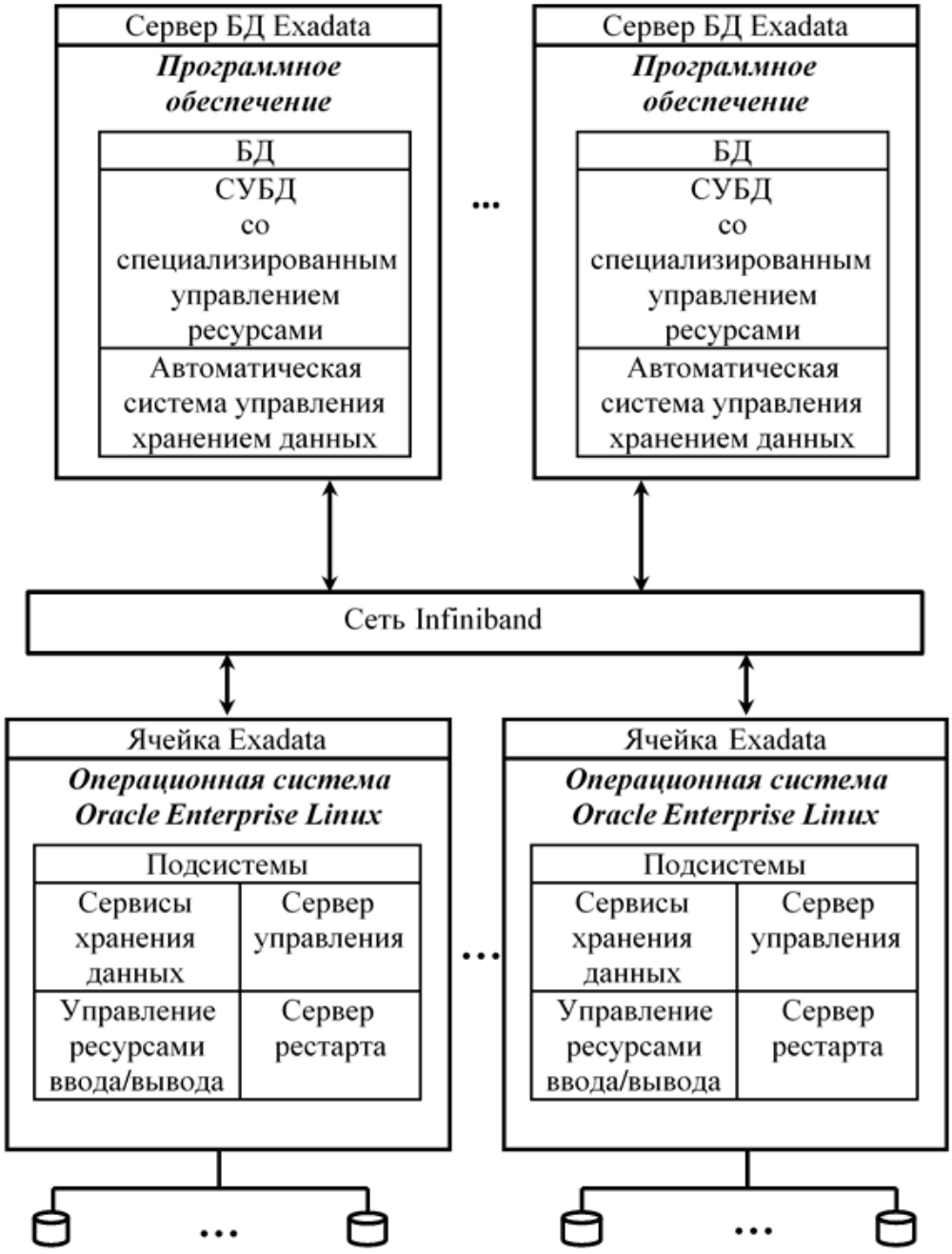


Рис. 1.6. Архитектура машины баз данных Oracle Exadata

Расположенная на сервере хранения данных СУБД Oracle имеет в своем составе подсистему управления ресурсами (Database Resource Manager (DBRM)), которая позволяет управлять распределением пропускной способности канала как между несколькими БД, так и в рамках одной БД, управлять выделением ресурсов процессора и параллельным выполнением операций.

Для управления системой хранения данных в ячейке Exadata используется автоматическая система управления (Oracle Automatic Storage Management (ASM)). Она используется как файловая система и управляет томами данных в ячейке Exadata. ASM обеспечивает виртуализацию ресурсов хранения данных, равномерно распределяя файлы базы данных по всем доступным ячейкам Exadata.

Основной компонент программного обеспечения ячейки Exadata – подсистема CELLSRV (Cell Services), которая реализует большинство сервисов хранения и обработки данных. Это многопоточный программный комплекс, который взаимодействует с БД на сервере и передает блоки данных по протоколу "интеллектуальная база данных" (Intelligent Database (iDB)). В функции CELLSRV входит просмотр блоков данных с целью определения столбцов и строк, которые удовлетворяют SQL-запросам.

Операции ввода/вывода реализуются прикладным программным обеспечением ячейки Exadata и регулируются менеджером ресурсов ввода/вывода (I/O Resource Manager (IORM)).

Сервер управления (Management Server (MS)) реализует функции администрирования, управления и контроля состояния ячейки Exadata.

Сервер перезагрузки (Restart server (RS)) поддерживает непрерывное функционирование программного обеспечения Exadata и используется для его обновления.

Интеграция функций СУБД с аппаратным обеспечением системы хранения позволяет эффективно выполнять операции над данными. Внедрение этих функций практически на аппаратный уровень позволяет существенно ускорить выполнение операций над данными и повысить эффективность обработки дан-

ных в целом. В результате запросы, которые сканируют таблицы, выполняются в ячейках Exadata, где происходит фильтрация строк, столбцов, соединение таблиц и другие действия. Серверу возвращаются только результаты запроса.

Нельзя обойти вниманием методы и алгоритмы, предложенные при создании прототипа параллельной СУБД Омега на базе многопроцессорного вычислительного комплекса МВС-100/1000. Разработанные методы организации параллельного выполнения запросов и балансировки загрузки применительно к гибридной архитектуре, используемой для построения высокоэффективных, масштабируемых, отказоустойчивых параллельных систем баз данных, показали высокую эффективность [96-98].

Рассмотренные программно-аппаратные вычислительные комплексы в ходе многочисленных экспериментальных проверок показали, что при их использовании многократно возрастает эффективность обработки данных. Вместе с тем, поскольку это комплексы общего назначения, предназначенные для решения различных классов задач, при их проектировании в полной мере не учитывались особенности каждого класса. Поэтому для решения задач из конкретных предметных областей, создавались и создаются частные технологии. Типичные примеры таких технологий – MapReduce и Hadoop, эффективно используемые для решения поисковых задач для неструктурированных данных.

Использование особенностей МОД, рассмотренных в 1.1, позволит построить специфические модели данных, на основе которых будет построена технология, обеспечивающая дополнительные возможности для повышения эффективности решения задач на программно-аппаратных вычислительных комплексах, подобных рассмотренным в этом параграфе. Построению моделей посвящены главы 2 и 3.

В качестве архитектурных решений целесообразно рассмотреть два подхода к параллельной реализации задач МОД:

1. в потоковой модели вычислений на основе теоретико-множественной модели из главы 2 и, в том числе, архитектуры вычислительных систем с ассоциативным распределением ресурсов [99, 100];

2. в матричной модели вычислений с обобщением параллельных алгоритмов, реализующих операции над плоскими матрицами, на многомерные матрицы [101, 102].

Из рассмотренных в 1.1 примеров и анализа в 1.2 архитектур вычислительных комплексов и их промышленных образцов видно, что реализация МОД возможна на различных вычислительных комплексах, от простейших вычислительных машин и сетей до сложных многопроцессорных комплексов, реализующих SMP и MPP архитектуру. В главах 4, 5, 6 показано, что использование предложенных в работе моделей и методов, а также способов организации данных и доступа к ним (например, индексно-последовательного) может повысить эффективность рассмотренных в этом параграфе вычислительных комплексов при решении задач МОД.

1.3. Проблемы оптимизации процессов массовой обработки данных

Проблема оптимизации обработки данных возникла одновременно с вычислительными машинами. Вначале программисты боролись за уменьшение объема используемой памяти и времени выполнения программы. Развитие вычислительной техники сделало память менее критичным ресурсом, но время выполнения программного комплекса, решающего прикладную задачу, попрежнему критично. Если на ранних этапах использования вычислительной техники временные характеристики программного обеспечения целиком зависели от искусства программиста, то появление алгебраических моделей данных, таких как реляционная, создало предпосылки для решения задач повышения эффективности обработки данных двумя способами:

– преобразованием имеющегося алгебраического выражения запроса с целью повышения эффективности его обработки;

– синтезом на основе спецификации (постановки) задачи эффективного алгебраического выражения запроса.

В обоих случаях достижение основного результата связано с максимальным использованием свойств программного обеспечения СУБД и архитектуры вычислительного комплекса.

Первые решения проблемы оптимизации запросов к базам данных основывались на следующих посылках.

В современных системах баз данных прикладному программисту доступно логическое представление данных, например, табличное или многомерно-матричное, не зависящее от того, как на самом деле реализованы хранение данных и доступ к ним на конкретном вычислительном комплексе (машине). Прикладной программист или неквалифицированный пользователь при создании запроса ничего не знают (и не должны знать) о том, как он будет реализован СУБД. Поэтому он не может нести ответственность за степень эффективности реализации запроса.

Выполнение сложных запросов на больших объемах данных требует много машинного времени. Но это время не должно зависеть от формулировки запроса. Намерения пользователя должны быть сохранены, но детали запроса могут быть преобразованы СУБД для обеспечения быстрой реакции. Это позволит защитить пользователей от катастрофически дорогих запросов. В то же время опытный программист может выразить свои запросы так, что они не потребуют оптимизации. В этом случае СУБД не должна обременять его дополнительными затратами времени на оптимизацию. То есть выгода от оптимизации возможна только в том случае, когда сложные запросы сформулированы неудачно и на их выполнение потребуется слишком много машинного времени [103].

Решение задач оптимизации запросов привело, в конечном счете, к разработке стандартного процесса, посредством которого различные СУБД исполняют запросы [104, 105]:

1. Запрос, написанный на языке манипулирования данными, подвергается синтаксическому анализу и преобразуется в дерево разбора, представляющее структуру запроса в виде, удобном для дальнейшего использования.

2. Дерево разбора преобразуется в дерево выражений некоторой алгебры (модели данных), которое называют логическим планом запроса.

3. Логический план запроса преобразуется в физический план запроса, который фиксирует отдельные операции, регламентирует порядок их выполнения, задает алгоритмы, реализующие операции, определяет способы получения данных и информационного обмена.

В ходе реализации этого процесса на этапе преобразования логического плана в физический план производится оптимизация запроса. Оптимизация возможна, поскольку в различных моделях данных алгебраические операции имеют способствующие этому свойства. А именно:

– Бинарные операции в большинстве случаев ассоциативны, то есть $(A\omega(B\omega C))= ((A\omega B)\omega C)$, где A, B, C – агрегаты данных, а ω – бинарная операция. Среди возможных исключений, например, теоретико-множественная операция разности (в МОД – слияние строго упорядоченных файлов).

– Бинарные операции, как правило, коммутативны, то есть $(A\omega B = B\omega A)$, где A, B – агрегаты данных (файлы, отношения), а ω – бинарная операция. Среди возможных исключений, например, теоретико-множественная операция, выполняющая действие подобное тому, которое выполняет операция слияния строго упорядоченных файлов в МОД, при реализации корректировки данных.

– Все унарные операции дистрибутивны относительно бинарных операций $(\omega_1(A\omega_2 B))= ((\omega_1 A)\omega_2(\omega_1 B))$, A, B – агрегаты данных, ω_1 – унарная операция, ω_2 – бинарная операция.

– Мультипликативная бинарная операция дистрибутивна относительно аддитивной бинарной операции $(A\omega_1(B\ \omega_2 C))= ((A\ \omega_1 B)\omega_2(A\omega_1 C))$, A, B, C – агрегаты данных, ω_1 – мультипликативная бинарная операция, ω_2 – аддитивная бинарная операция.

На основе этих законов строятся стратегии оптимизации планов запросов, реализующие следующие действия:

– продвижение унарных операций выборки и проекции (*select* и *project*) как можно ниже по дереву;

– сочетание унарных операций с бинарными, то есть замена двух операций одной;

– построение цепочек бинарных операций, время выполнения которых минимально.

Для реализации этих стратегий используются различные методы оптимизации, например, динамическое программирование или методы, основанные на эвристических алгоритмах (метод ветвей-границ, жадные алгоритмы) [72, 105].

Рассмотренные методы повышения эффективности обработки данных широко используются в современных СУБД, таких как Microsoft SQL Server, Oracle, IBM DB2 и других [106-110].

Некоторые запросы имеют регулярный характер, то есть выполняются не один раз, а многократно. При каждом выполнении такого запроса последовательность действий, приводящих к получению результата, не меняется, изменяются только входные параметры, не влияющие на характер и последовательность операций процесса обработки данных. Типичный пример такого запроса – запрос баланса пластиковой банковской карты, который передается банкоматом серверу баз данных. Для ускорения выполнения таких запросов используются хранимые процедуры – объекты базы данных, представляющие собой наборы SQL-инструкций, которые компилируются и оптимизируются один раз, после чего постоянно хранятся на сервере. В хранимых процедурах могут выполняться стандартные операции с базами данных, определенные в языках управления и манипулирования данными (DDL и DML). Кроме того, в них могут использоваться инструкции управления процессом исполнения, обеспечивающие возможность организации циклов и ветвлений. Хранимые процедуры обычно создаются с помощью языка SQL и конкретной его реализации в выбранной СУБД [111]. Например, для этих целей в СУБД Microsoft SQL Server используется язык Transact-SQL, в Oracle – PL/SQL, в IBM DB2 – SQL/PL в PostgreSQL – pgSQL.

Рассматриваемому в работе варианту МОД присущи именно регулярные запросы. Такие запросы разрабатываются прикладным программистом в про-

цессе проектирования автоматизированной информационной системы, после чего существуют практически без изменений на протяжении всего жизненного цикла системы. Следовательно, затраты на трансляцию запроса, его оптимизацию, создание библиотечной программы, аналогичной хранимой процедуре, имеют разовый характер и не наносят ущерба как разработчику – профессиональному прикладному программисту – так и самой информационной системе во время ее эксплуатации. Методы синтеза оптимизации запросов – процессов массовой обработки данных – рассмотрены в главе 5.

Особенность современного состояния вычислительной техники состоит в том, что, во-первых, используются гибкие архитектуры современных аппаратных средств, основанные на многоядерности и многопроцессорности, а во-вторых, вычислительные сети обеспечивают такую простоту коммуникаций, которая позволяет легко проектировать различные топологии сетей. Эти два фактора, – регулярность запросов и гибкость вычислительных средств, – позволяют решать задачи оптимизации архитектуры программно-аппаратного вычислительного комплекса на этапе разработки автоматизированной информационной системы. Причем для каждой системы, входящей в ее состав задачи или даже отдельного запроса, может быть разработан индивидуальный программно-аппаратный вычислительный комплекс.

Такой подход полностью соответствует идеям, предложенным в [43]. Он заключается в том, что "систему математического обеспечения ЭВМ следует разрабатывать одновременно с ее проектированием. В связи с этим актуальной проблемой современной вычислительной техники является автоматизация разработки систем математического обеспечения и проектирования ЭВМ как единого процесса. Ее решение требует развития новых теоретических направлений в кибернетике, в частности прикладной теории алгоритмов. Одним из основных источников задач прикладной теории алгоритмов является проблема оптимального перевода с одного языка на другой, которая может быть сформулирована следующим образом: существуют два алгоритмических языка и некоторый алгоритм, написанный на одном из них; требуется найти оптимальную по задан-

ным критериям реализацию этого алгоритма на другом языке. В программировании обычно первым является некоторый язык программирования, ориентированный на тот или иной круг задач, а вторым – внутренний язык машины, на которой решаются данные задачи. Таким образом, речь идет о трансляции с языка программирования на машинный язык с одновременной оптимизацией выходной программы. В то же время исходным может быть алгоритм работы некоторого устройства ЭВМ, записанный на предназначенном для этой цели алгоритмическом языке, а язык, на который транслируется данный алгоритм, – это язык схем. Тогда задача состоит в получении оптимальной схемы, реализующей алгоритм работы данного устройства или некоторой его части. Процесс решения таких задач на практике делится на промежуточные этапы, на каждом из которых выполняется некоторая частичная оптимизация исходного алгоритма. Каждому из этих этапов соответствует свой промежуточный язык, причем перевод с одного промежуточного языка на другой должен осуществляться достаточно просто. Тогда оптимизацию можно проводить с помощью эквивалентных преобразований алгоритма, полученного на данном этапе с учетом его последующей трансляции на язык очередного этапа. Для выполнения тонких и глубоких эквивалентных преобразований алгоритмов необходимо построить алгебру, которая позволила бы производить эквивалентные преобразования столь же простым и естественным способом, каким они выполняются в обычной алгебре или анализе".

Все сказанное в приведенной цитате полностью относится к реализации МОД. Как было показано в разделе 1.2.2, современные вычислительные комплексы, ориентированные на массовую обработку данных, позволяют довести программное обеспечение до уровня аппаратуры, используя "микропрограммы", которые размещаются в ПЛИС (FPGA). То есть, становится возможной трансляция с языка, на котором написаны алгоритмы реализации операций, на язык схем.

Таким образом, в задачах МОД можно выделить две основные цели оптимизации:

– ускорение отдельных операций обработки одного или нескольких файлов,

– построение оптимальных последовательностей операций (процессов или запросов) для решения конкретных задач.

Для достижения этих целей необходима разработка математических моделей, с помощью которых станет возможной формализация всех составляющих логически последовательной обработки данных: записей, файлов, операций и процессов обработки файлов. На основе этих моделей становятся возможными:

– оптимизация отдельных операций за счет применения различных параллельных архитектур вычислительных комплексов, реализующих выполнение операций;

– синтез и оптимизация процессов (запросов), составленных из этих операций обработки данных, с одновременным выбором наилучших архитектур вычислительных комплексов, реализующих эти процессы.

То есть, предложенные модели позволят производить двухуровневую оптимизацию, как на уровне операций, так и на уровне процессов.

Как было сказано, такие модели рассматриваются в главе 2, а в главах 4 и 5 рассматриваются методы оптимизации процессов и выбора архитектур вычислительных комплексов, эффективно реализующих эти процессы.

1.4. Требования к моделям массовой обработки данных

Теоретическую основу работы составляют две алгебраические модели МОД. Это теоретико-множественная или файловая и многомерно-матричная модели данных.

Известно, что проектирование БД многоступенчатый процесс, состоящий из последовательного построения моделей данных различных уровней.

Первый уровень – концептуальное (инфологическое) проектирование, – заключается в построении семантической или концептуальной модели предметной области, то есть информационной модели наиболее высокого уровня

абстракции. Такая модель есть образ предметной области и прообраз проектируемой для нее базы данных.

Второй уровень – логическое (даталогическое) проектирование, – состоит в создании схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. В реляционной модели данных даталогическая модель реализована набором схем отношений, с указанием первичных ключей, идентифицирующих кортежи каждого отношения и внешних ключей, задающих связи между отношениями. Выбор модели данных, во многом, определяется концептуальной моделью.

Третий уровень – физическое проектирование, – реализует создание схемы базы данных средствами конкретной СУБД, реализующей выбранную модель данных. Специфика конкретной СУБД при физическом проектировании определяет набор решений, связанных с моделью вычислений и физической средой хранения данных, а именно: выбор методов доступа к данным, управления внешней памятью, распределения всей БД или отдельных таблиц по файлам и устройствам.

Вместе с тем, ни одна модель данных не может быть ориентирована на все возможные архитектуры вычислительных комплексов, которые в современных условиях доступны для использования разработчикам БД. Кроме того, промышленные СУБД ориентированы на конкретные вычислительные машины и системы (платформы). При этом не учитывается тот факт, что гибкость современных платформ позволяет моделировать на их основе различные архитектуры вычислительных комплексов. Попытка учесть такую возможность, несомненно, привела бы к неоправданному усложнению программного обеспечения СУБД. Поэтому целесообразно предоставить прикладному программисту формальный аппарат в виде дополнительных моделей данных. Они должны обеспечить связь между моделью данных, выбранной для формализации решаемой задачи, СУБД, выбранной для реализации алгоритмов, решающих задачу, и архитектурой вычислительного комплекса, который наилучшим образом обеспечивает решение задачи. В результате происходит объединение вычислительных

средств, прикладных программ и программного обеспечения СУБД, которые должны обеспечить решение конкретной задачи, в единый программно-аппаратный комплекс.

Как было сказано, в работе, в качестве таких связующих моделей предложены теоретико-множественная, или файловая, и многомерно-матричная модели данных (главы 2, 3). Далее перечисляются требования, которым эти модели должны удовлетворять.

1. Соответствие моделям данных и вычислений. Неформально, требование соответствия двух моделей означает наличие у них свойств, позволяющих использовать одну модель МОД вместо другой. В работе рассматриваются два типа соответствия моделей. Первый тип состоит в том, что:

– каждому набору данных (прообразу), представленному в одной модели (например, отношению в реляционной, файлу в теоретико-множественной), ставится в соответствие один и только один набор данных (образ), представленный в другой модели (например, многомерная матрица);

– каждой операции в одной модели ставится в соответствие одна и только одна операция или композиция операций в другой модели, и результату операции над прообразами соответствует результат операции над образами.

В математике, в частности, в теории абстрактных алгебраических систем, такое соответствие называется изоморфизмом. Второй тип соответствия состоит в том, что:

– каждому набору данных (прообразу), представленному в одной модели, ставится в соответствие единственный набор данных (образ), представленный в другой модели (например, логическая многомерная матрица), то есть, нескольким прообразам может соответствовать один и тот же образ;

– каждой операции в одной модели ставится в соответствие одна и только одна операция или композиция операций в другой модели, и результату операции над прообразами соответствует результат операции над образами.

Такое соответствие называется гомоморфизмом.

2. Процедурность. Модель должна обеспечивать алгебраическую процедурную формулировку запроса, которая задает правила его реализации. То есть запрос, представленный на языке модели, может быть вычислен на основе выполнения элементарных алгебраических операций, определенных в модели, с учетом приоритетности, возможного наличия скобок и некоторых дополнительных правил, определяющих порядок их выполнения. К требованию процедурности рассматриваемых в работе моделей добавляются два дополнительных требования.

Первое требование состоит в том, что и сами элементарные алгебраические операции, которые реализуют выполнение запроса, должны иметь формализованные описания, позволяющие проектировать процедуры, реализующие их алгоритмы, таким образом, чтобы они наилучшим образом выполнялись в используемой модели вычислений. В распространенных в настоящее время моделях данных эти операции не имеют формализованных описаний. Поэтому качество их реализации определяется мастерством программистов, которые разрабатывают СУБД.

Второе требование заключается в предоставлении прикладному программисту возможности применения способов организации и распределения данных, не реализованных в конкретной СУБД, и разработки на основе имеющегося языка манипулирования данными процедур запросов, эффективно использующих выбранные организацию данных и модель вычислений.

Применение моделей данных, соответствующих этим требованиям позволит:

– программистам, разрабатывающим СУБД, – расширить круг архитектур вычислительных комплексов, на которых станет возможным применение этих СУБД;

– прикладным программистам – возможность привязывать различные СУБД к выбранной для решения прикладной задачи архитектуре вычислительного комплекса.

3. Параллелизм алгебраических операций. Формализация операций должна обеспечивать возможность распараллеливания операций совместной обработки

двух или более файлов. В главе 5 показано, что формальное определение операций в теоретико-множественной модели позволяет использовать методы распараллеливания, присущие системам, управляемым данными (data flaw), а в многомерно-матричной модели – системам на основе векторных (матричных) процессоров. Кроме того, модель должна обеспечивать возможность распараллеливания обмена между оперативной памятью и внешней запоминающей средой (традиционными внешними носителями информации или хранилищами данных).

4. Оптимизация запросов. Модель должна обеспечивать возможность оптимизации процессов совместной обработки нескольких файлов, реализующих запросы. Это означает, что в терминах модели процесс, реализующий запрос, должен иметь формальное представление в виде алгебраического выражения, которое можно либо автоматически синтезировать с заданными характеристиками, либо преобразовывать для улучшения его характеристик. Спецификацией для построения выражения может служить неформальное описание запроса, например, это может быть набор, содержащий описания входных, данных, правил преобразований атрибутов (формул для вычисления значений) и результата. Формальное описание запроса на языке любой модели данных, например, SQL-модели, также может служить спецификацией для синтеза выражения, или его оптимизации в процессе трансляции на язык связующей модели. Следовательно, связующая модель должна содержать средства, с помощью которых возможно реализовать синтез нового оптимального процесса и оптимизацию имеющегося процесса посредством эквивалентных преобразований, оптимизировать имеющийся процесс. Поскольку методы оптимизации процесса, как правило, имеют высокую вычислительную сложность, в большинстве случаев оперативная оптимизация может быть затруднительной. Поэтому целесообразно оптимизировать многократно выполняющиеся процессы. Вместе с тем, параллельная реализация некоторых методов оптимизации позволяет надеяться на возможность оперативной оптимизации процессов. Проблема оптимизации

запросов имеет давнюю историю, но в работе рассматриваются методы, учитывающие особенности не только модели данных, но и модели вычислений.

5. Объектно-ориентированная парадигма. Это требование означает, что рассматриваемые в работе формальные модели должны обеспечивать возможность применения современных объектно-ориентированных методов проектирования и программирования. Вместе с тем, принятый сегодня подход к описанию объектных моделей данных имеет сугубо технологический характер [71, 72, 112], ведущий свою родословную от первых работ в этой области [113]. Технологический характер выражается в том, что отсутствует строгое определение объекта, которое заменяется неформальным описанием его свойств и свойств систем баз данных, построенных на основе объектно-ориентированного подхода. Это приводит к тому, что даже активные сторонники и основоположники объектного подхода к построению систем баз данных указывают на его недостаток, состоящий в отсутствии строго определения объекта, то есть "объект – это все, что угодно" [114]. Рассматриваемые в работе модели будут построены именно как объектные, на основе предложенного в [53] строгого определения абстрактного типа данных (объекта, класса) как универсальной многоосновной алгебраической системы.

Перечисленные в этом параграфе требования будут учтены при построении объектно-ориентированных теоретико-множественной и многомерно-матричной моделей в главе 2.

1.5. Заключительные замечания к главе 1

В этой главе дано определение МОД как способа параллельной обработки больших объемов данных большим числом процессоров. Указано, что в работе рассматривается и исследуется один из видов МОД – обработка структурированных данных.

Приведены примеры задач этого класса и сделан вывод о том, что при их решении в обработку включаются практически все данные, характеризующие объекты этих задач, и объемы обрабатываемых данных очень велики.

Проведен анализ архитектур известных промышленных программно-аппаратных комплексов IBM Netezza и Oracle Exadata с позиции использования их для реализации МОД.

Определены цели работы, состоящие в разработке моделей данных, обеспечивающих наибольшее соответствие существующим моделям данных и архитектурам вычислительных комплексов, и разработке на основе этих моделей способов организации данных во внешней памяти и алгоритмов реализации операций, в наибольшей степени соответствующих структурам данных.

Введено понятие коэффициента активности данных и на его основе определены условия, при которых целесообразно использование методов МОД.

Приведено неформальное описание операций над файлами в МОД. Определено понятие логически последовательного метода доступа к данными и выбраны, на основе известных классификаций, архитектуры программно-аппаратных комплексов, наилучшим образом реализующие этот метод доступа.

Для решения задачи формализации процессов МОД предложены теоретико-множественная, или файловая, и многомерно-матричная модели данных и сформулированы требования, которым эти модели должны удовлетворять.

В число требований вошли: соответствие моделям данных и вычислений; процедурность; параллелизм алгебраических операций; оптимизация запросов; объектно-ориентированная парадигма. Основные результаты, полученные в данной главе, были опубликованы в работах [59, 63, 69, 73, 74, 85, 86].

Глава 2. АЛГЕБРАИЧЕСКИЕ МОДЕЛИ ДАННЫХ ДЛЯ ПОСТРОЕНИЯ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ

2.1. Основные алгебраические понятия

2.1.1. Универсальные алгебры и алгебраические системы

В качестве основы объектно-ориентированных (алгебраических) моделей, которые позволят формализовать представления данных и операции их преобразования, используемые при обработке файлов, в дальнейшем используются такие понятия как универсальная алгебра и универсальная алгебраическая система, известные определения которых [115, 116] приведены ниже.

Определение 2.1. Пусть A – некоторое непустое множество. Частично определенная функция $y = F(x_1, \ldots, x_n)$, $(y, x_1, \ldots, x_n) \in A$, называется n-арной частичной операцией на A. Если функция всюду определена, говорят просто об n-арной операции.

Определение 2.2. Система $U_A = < A, \Omega >$, состоящая из основного множества A и определенной на нем совокупности частичных операций $\Omega = \{F_s^{n_s}\}(s = 1, 2, \ldots)$, называется частичной универсальной алгеброй с сигнатурой Ω.

Определение 2.3. Универсальные алгебры U_A и U_B, в которых заданы соответственно сигнатуры Ω и Ω', называются однотипными, если можно установить такое взаимно-однозначное соответствие между сигнатурами Ω и Ω', при котором любая операция $F \in \Omega$ и соответствующая ей операция $F' \in \Omega'$ будут n-арными с одним и тем же n.

Пусть даны две однотипные универсальные алгебры $U_A = < A, \Omega >$ и $U_B = < B, \Omega >$ с основными множествами A и B.

Определение 2.4. Отображение $\varphi : A \to B$ называется гомоморфным отображением алгебры U_A в алгебру U_B, если для любых элементов $a_1, \ldots, a_n \in A$ и произвольной n-арной операции $F \in \Omega$ выполняется соотношение

$\varphi(F(a_1, \ldots, a_n) = F(\varphi(a_1), \ldots, \varphi(a_n)$, где $\varphi(a_i) = b_i$ и $b_i \in B(i = 1, ..., n)$. Алгебры U_A и U_B называются гомоморфными.

Если между основными множествами A и B устанавливается взаимно-однозначное соответствие, то отображение φ называется изоморфным отображением, а алгебры U_A и U_B называются изоморфными.

Определение 2.5. Пусть $\pi(x_1, ..., x_n)$, $x_1, ..., x_n \in A$ – n-местный предикат, $\Pi = \{\pi_s^{n_s}\}$ $(s = 1, 2, \ldots)$ – сигнатура предикатов. Система $\boldsymbol{U}_A$=<A; Ω; П> называется универсальной алгебраической системой.

Между универсальными алгебраическими системами также можно устанавливать гомоморфные и изоморфные отображения. Для этого необходимо установить соответствие и между сигнатурами предикатов, подобное тому, которое устанавливается между сигнатурами операций.

Случаи, когда при построении формальной модели решаемой задачи можно обойтись единственным основным множеством, встречаются нечасто, поскольку объекты предметной области имеют сложную структуру, для описания которой используется много разнотипных параметров. Для моделирования таких предметных областей используются многоосновные алгебры.

Определение 2.6. Система $U_{\boldsymbol{M}}$ =< $\boldsymbol{M}$; Ω>, состоящая из семейства основных множеств $\boldsymbol{M}$={A_α} (α = 1, 2, …) и сигнатуры Ω операций, определенных на семействе $\boldsymbol{M}$ так, что каждая n-арная операция из Ω является отображением декартова произведения n множеств из семейства $\boldsymbol{M}$ в множество из того же семейства $A_{\alpha_1} \times \ldots \times A_{\alpha_n} \to A_{\alpha_r}$, называется многоосновной алгеброй.

Определение 2.7. Система $\boldsymbol{U}_{\boldsymbol{M}}$=<$\boldsymbol{M}$; Ω; П>, где П – сигнатура n-местных предикатов $\pi : A_{\alpha_1} \times \ldots \times A_{\alpha_n} \to \{0, 1\}$, называется многоосновной алгебраической системой.

Между многоосновными алгебрами и алгебраическими системами, также, как и между одноосновными, можно устанавливать гомоморфные и изоморфные отображения.

Многоосновные алгебры и алгебраические системы играют важную роль в программировании. Они представляют собой теоретическую базу для создания технологий построения пользовательских типов данных в современных языках программирования. А некоторые встроенные в языки типы данных сами есть не что иное, как многоосновные алгебраические системы. Далее приводятся примеры некоторых часто встречающихся типов данных, которые могут быть представлены в виде многоосновных алгебраических систем [117].

Пример 2.1. Пусть $\boldsymbol{U_S}$=<S, Z_0; Ω; П>– система, состоящая из множества строк S и множества неотрицательных целых чисел Z_0. Сигнатура Ω состоит из операций:

$\textcircled{L}: S \to Z_0$	– длина строки;
$\textcircled{P}: S \times S \to Z_0$	– позиция подстроки s_1 в строке s_2;
$\textcircled{+}: S \times S \to S$	– конкатенация (сцепление) строк s_1 и s_2;
$\textcircled{C}: S \times Z_0 \times Z_0 \to S$	– выделение из строки s_1, начиная с позиции i, подстроки s_2 длины l.

Сигнатура предикатов П может включать предикаты:

- π_1: "Строка s – пустая" (не содержит ни одного символа),
- π_2: "Строка s_1 предшествует строке s_2".

Таким образом, строковый тип в языках программирования – это двухосновная универсальная алгебраическая система.

Пример 2.2. Обычная алгебра матриц представляет собой очень важный с точки зрения массовой обработки данных пример универсальной двухосновной алгебраической системы $\boldsymbol{U_M}$=<M, R;Ω; П>, где M – множество матриц, а R– множество действительных чисел(элементов матриц). Сигнатура Ω универсальной двухосновной алгебраической системы $\boldsymbol{U_M}$ имеет следующий вид:

$+: R \times R \to R$	– аддитивная операция над элементами матриц;
$\times: R \times R \to R$	– мультипликативная операция над элементами матриц;
$\textcircled{t}: M \to M$	– транспонирование матрицы;

$\times : R \times M \to M$	– умножение матрицы на число;
$+ : M \times M \to M$	– сумма матриц;
$\times : M \times M \to M$	– произведение матриц;
$\textcircled{D} : M \to R$	– определитель матрицы.

Сигнатура предикатов П может содержать такие предикаты как "матрица M вырожденная" или "матрица M_1 может быть умножена на матрицу M_2".

Далее многоосновные алгебраические системы будут использованы в качестве аппарата для построения моделей данных, свойства которых удовлетворяют поставленным в работе задачам.

2.1.2. Интуитивный подход к объектно-ориентированному моделированию, проектированию и программированию

Основу объектно-ориентированного подхода составляет понятие абстрактного типа данных (АТД) [113]. В основном, в современной литературе рассматривается интуитивный подход к описанию АТД, и на его основе строятся технологии объектно-ориентированного моделирования, проектирования и программирования.

Определение 2.8. Интуитивно АТД – это конструкция языка программирования, в которой группируются в одно понятие следующие элементы:

– набор операций (действий),

– множество (одно или более) объектов, к которым эти операции применяются,

– защита, или возможность средствами языка и соответствующей ему системы программирования защитить внутреннее представление АТД от действий, не указанных явно в его определении (то есть скрыть его от пользователя АТД) [118].

Определение 2.9. В языках программирования АТД – это конструкция, состоящая из двух частей:

– интерфейса, содержащего имя определяемого АТД, имена операций с указанием типов их аргументов и значений;

– описания операций и объектов, с которыми эти операции работают, средствами обычного языка программирования.

Это описание называется конкретным, в отличие от абстрактного описания, которое делается средствами более высокого уровня. Конкретное описание также называется реализацией или представлением АТД. Благодаря защите только имена, перечисленные в интерфейсе, доступны, то есть могут использоваться другими компонентами программы, внешними по отношению к АТД.

В дальнейшем при конструировании моделей данных будут использованы алгебраическое определение АТД и конкретные описания для каждого АТД. Интерфейс АТД представляет собой перечисление переменных, принимающих значения на основных множествах АТД, и операций, определенных на этих множествах и принимающих значения в одном из них. В реализации АТД переменным присваиваются имена и определяются их типы. Операциям ставятся в соответствие программные реализации (процедуры и функции), состоящие из двух частей:

– декларативной, в которой каждой функции, реализующей операцию, присваивается имя и приводится описание ее интерфейса (задается список формальных параметров);

– императивной, являющейся телом подпрограммы, реализующей эту функцию.

В одних языках программирования (ObjectPascal, C++), декларативная и императивная части реализации операции отделены друг от друга и находятся в разных разделах программного кода или даже в разных файлах, в других (C#) – совмещены. В современных объектных технологиях принято называть переменные свойствами объекта, а функции, реализующие операции – методами объекта.

Для того чтобы разработка: модель, проект, программа, соответствовала требованиям объектно-ориентированной технологии, необходимо, чтобы каждый АТД обладал следующими свойствами:

Абстракция – отображение в АТД только значимых характеристик и свойств моделируемого объекта.

Инкапсуляция – объединение данных и операций их обработки и сокрытие деталей реализации от пользователя; то есть, иными словами, инкапсуляция – это защита данных и операций (свойств и методов) АТД от несанкционированных действий пользователя.

Наследование – возможность использования одним АТД свойств и методов другого. АТД, от которого производится наследование, называется базовым (родительским), а наследующий АТД – наследником (производным). В языках программирования существует два метода организации наследования: прямое указание того, что новый АТД есть наследник имеющегося базового, и косвенное (контейнерный метод), реализуемое включением базового АТД в производный в качестве одного из свойств.

Полиморфизм – возможность использовать одинаковые знаки для операций (в языках программирования – имена функций, реализующих операции), которые имеют одинаковый или схожий смысл, но выполняются над разнотипными данными, и, следовательно, реализованы разными алгоритмами. Например, знак плюс используется для обозначения операции сложения как действительных, так и комплексных чисел, или, как в примере 2.2, знак + использовался как для обозначения сложения матриц, так и для сложения их элементов, а знак × – для умножения двух матриц, матрицы на число и элементов матриц.

Интуитивное определение АТД оказалось весьма продуктивным, поскольку позволило создать на его основе как технологии объектно-ориентированного программирования, так и технологии моделирования данных и проектирования баз данных [119-123]. Однако для решения поставленных в работе задач связывания моделей данных и моделей вычислений методом установления соответствия между этими моделями интуитивного определения АТД недостаточно. Поэтому необходимо ввести и использовать строгое определение АТД; рассмотрению его и следующих из него методов построения моделей данных посвящены следующие разделы этой главы.

2.1.3. Алгебраический (формальный) подход к объектно-ориентированному моделированию, проектированию и программированию

Из интуитивного определения следует, что АТД объединяет под одним именем наборы данных и операций над ними, при помощи которых можно задать описание свойств и процедур преобразования реального объекта из некоторой предметной области, то есть построить его формальную модель. Поэтому в ряде работ, например, [124], АТД определяется как математическая модель с совокупностью операторов, определенных в рамках этой модели.

Для строгого определения АТД используется понятие многоосновной алгебраической системы.

Определение 2.10. Абстрактный тип данных (он же объект или класс)– это универсальная одноосновная или многоосновная алгебраическая система [53].

Это определение позволяет рассматривать в качестве АТД любые, в том числе и одноосновные, универсальные алгебры и алгебраические системы. Оно, несмотря на краткость, полностью соответствует интуитивному определению [125]. Следующий пример это наглядно иллюстрирует.

Пример 2.3. При решении различных вычислительных и логических задач, таких как: поиск кратчайших путей, определение доступности вершин графа [126], разузлование [127], для определения данных часто используется универсальная одноосновная алгебраическая система – моноид.

Моноидом называется множество $\boldsymbol{M}$, на котором задана бинарная операция (∗), и выполняются два условия:

1. для любых трех элементов $x \in \boldsymbol{M}$, $y \in \boldsymbol{M}$, $z \in \boldsymbol{M}$ $(x*(y*z))=((x*y)*z)$ – ассоциативность операции;
2. существует такой элемент $\boldsymbol{e} \in \boldsymbol{M}$, называемый нейтральным элементом, что для любого $x \in \boldsymbol{M}$, $x * \boldsymbol{e} = x = \boldsymbol{e} * x$.

Моноид обозначается как тройка вида $\boldsymbol{U_M} = <\boldsymbol{M}; *; \boldsymbol{e}>$. Определенный таким образом моноид можно рассматривать как базовый АТД, то есть наивысший уровень абстракции, которому соответствуют различные реализации, каждая из которых соответствует конкретной задаче или конкретному классу задач:

1. U_R=<R;+; 0> – множество действительных чисел с операцией сложения;
2. U_R=<R; ×; 1> – множество действительных чисел с операцией умножения;
3. $U_R =< R; \min; +\infty >$ – множество положительных действительных чисел с операцией нахождения минимального из двух чисел и нейтральным элементом ($+\infty$), которому в программных реализациях, как правило, соответствует наибольшее значение в типе.
4. U_B=<B; ∨; 0> –множество B={0, 1} с операцией дизъюнкции.
5. U_S=<S;+; ∅> – множество символьных строк произвольной длины с операцией конкатенации (сцепления) строк и нейтральным элементом "пустая строка" (строка, не содержащая ни одного символа);

Следовательно, АТД "Моноид" удовлетворяет требованию абстракции.

Свойство инкапсуляции удовлетворяется в силу того, что в каждой реализации базового АТД точно указано основное множество, а операция определена как алгебраическая, то есть замкнутая на основном множестве.

Все приведенные реализации моноида наследуют свойства, присущие базовому АТД, то есть свойство наследования также удовлетворяется.

Наконец, для операций двух реализаций моноида, показанных в пунктах 1 и 5 знак (+) используется для обозначения двух различных по смыслу операций (сложение действительных чисел и конкатенация строк). То есть удовлетворяется свойство полиморфизма операций.

Из следующего примера видно, что хорошо известный и широко применяемый в программировании класс (АТД) DataTable (таблица данных), принадлежащий платформе .NET Framework, может быть представлен как универсальная алгебраическая система.

Пример 2.4. Из множества всех операций (методов класса DataTable) [123] для построения примера универсальной многоосновной алгебраической системы выбрана только одна – метод Compute. Это обеспечивает простоту примера, но не ограничивает общность. Операция реализована функцией, которая определена на следующих множествах:

– множество всех таблиц T;

– множество строк специального вида S_E, содержащее выражения на основе агрегатных функций (сумма, среднее значение, максимум и им подобные);

– множество строк специального вида S_F, содержащее логические выражения – фильтры строк таблицы.

Поскольку тип результата операции определяется типом результата вычисления выражения из S_E, то для обеспечения общности он задается типом object. То есть результату можно присваивать значения любого типа, определенного в языке или определенного пользователем. Можно считать, что тип object соответствует понятию универсального множества $\boldsymbol{U}$, то есть множества, фиксированного в рамках решаемой задачи и содержащего в качестве элементов все объекты, рассматриваемые в этой теории. Тогда операцию можно определить как функцию $Compute: T \times S_E \times S_F \to \boldsymbol{U}$. Класс DataTable определяется как многоосновная универсальная алгебраическая система $U_{DT} =< T, S_E, S_F, \boldsymbol{U}; Compute; \Pi >$. Определение предикатов зависит от решаемой задачи. Например, "выражение фильтра принимает значение ***истина*** более чем на половине строк таблицы".

Примеры 2.3 и 2.4 явно демонстрируют соответствие двух определений АТД: интуитивного и строгого (алгебраического). Далее на основе строгого определения будет разработан базовый АТД, с помощью которого можно будет построить модели данных, позволяющие решать поставленные в работе задачи.

2.1.4. Объектно-ориентированный подход к разработке моделей данных

Для разработки моделей данных, удовлетворяющих требованиям, перечисленным в параграфе 1.3 (соответствие высокоуровневым моделям данных и моделям вычислений, процедурность, параллелизм алгебраических операций, возможность оптимизации запросов, объектная ориентированность), необходимо выбрать базовые АТД, свойства которых будут достаточными для того, чтобы, используя механизм наследования, можно было бы строить необходимые универсальные алгебры или алгебраические системы для использования их в качестве моделей данных.

Среди множества произвольных АТД предлагается рассмотреть специфический АТД, называемый в дальнейшем ***абстрактной алгебраической машиной*** [128-130].

Определение 2.11. Абстрактная алгебраическая машина – это двухосновная алгебраическая система вида $\boldsymbol{E}=<S, T; \Omega; \Pi>$. Основа S называется ***структурой***, а основа T – ***типом***.

Структура представляет собой некоторую конструкцию, составленную из экземпляров данного типа. Примеры такого рода структур – векторы, матрицы, графы. Выбор структуры и типа определяется особенностями решаемой задачи. Причем для некоторых классов задач одной структуре могут соответствовать несколько типов. Следующий пример иллюстрирует эту ситуацию.

Пример 2.5. Для решения названных в примере 2.3 задач, – поиск кратчайших путей, определение доступности вершин графа, разузлование, – может быть применен метод, основанный на алгоритме вычисления транзитивного замыкания квадратной матрицы M. Эта матрица задает отношение объектов некоторой предметной области: населенных пунктов и дорог, которые их соединяют, изделий и узлов и деталей, из которых они состоят, и тому подобных. Транзитивное замыкание матрицы M вычисляется по следующей формуле

$$M^* = \sum_{i=1}^{K} M^i,\ M^i \neq Z, \text{ для всех } i \leq K, \text{ и } M^{K+1} = Z, \text{где } Z - \text{нуль - матрица}.$$

В этом случае абстрактная алгебраическая машина имеет следующий вид $\boldsymbol{E_M}$=<*M*, *X*; Ω; П>, где *X*–тип, а *M* – множество квадратных матриц, составленных из элементов типа. Минимальное требование к типу *X* состоит в том, чтобы на *X* были определены две алгебраические операции, одна из которых трактуется как аддитивная, а вторая – как мультипликативная. То есть тип *X* должен быть по каждой из этих операций, по крайней мере, алгебраической структурой, называемой группоидом. В реальных задачах типами могут быть достаточно сложные алгебраические структуры, такие как кольца и поля. В таблице 2.1 приведены формальные определения и описания операций сигнатуры Ω абстрактной алгебраической машины $\boldsymbol{E_M}$.

Таблица 2.1. Сигнатура операций $\boldsymbol{E_M}$

Операция	Описание операции
$\oplus : X \times X \to X$	аддитивная операция над элементами матриц;
$\otimes : X \times X \to X$	мультипликативнаяоперация над элементами матриц;
$\textcircled{\prime} : M \to M$	транспонирование матрицы;
$+ : M \times M \to M$	сумма матриц;
$\times : M \times M \to M$	произведение матриц;
$\textcircled{D} : M \to X$	определитель матрицы.

В реальных задачах в роли типа *X* могут быть, такие множества как:

– в "задаче разузлования" – множество неотрицательных действительных чисел R^0, с аддитивной операцией сложения и мультипликативной операцией умножения чисел, Ω={+, ×};

– в задаче вычисления кратчайших путей в графе – множество положительных действительных чисел R^+, с аддитивной операцией вычисления минимума из двух чисел, и мультипликативной операцией сложения, Ω={*min*, +};

– в задаче определения доступности вершин в графе множество {0, 1} с аддитивной операцией дизъюнкции и мультипликативной операцией конъюнкции, Ω={∨, ∧}.

Операции над матрицами, входящие в сигнатуру операций Ω, реализуются хорошо известными последовательными и параллельными стандартными алгоритмами, о которых речь пойдет далее.

Важность рассмотренного примера состоит, прежде всего, в том, что он наглядно показывает наличие универсальных двухосновных алгебраических систем, обладающих качеством, которое можно сформулировать как возможность замены одного основного множества, а именно, типа и операций над его элементами, при сохранении другого множества и алгоритмов, реализующих операции над его элементами. Этот факт можно сформулировать в виде очевидного утверждения.

Утверждение 2.1. Пусть S – структура, а T_1, …, T_n – допустимые для этой структуры типы. Тогда T_1, …, T_n – однотипные универсальные алгебраические системы (определение 2.3).

Практическая ценность универсальных алгебраических машин состоит в том, что суть операций над элементами структуры S не изменяется при изменении сути операций над элементами типа T. Это свойство универсальных алгебраических машин может быть полезным в практическом программировании. Если типы T_1, …, T_n – гомоморфные или изоморфные универсальные алгебраические системы, то становится возможной отладка операций над структурой на наиболее простом типе данных. Отлаженная таким образом структура становится базовым АТД, от которого можно порождать конкретные АТД (реализации), предназначенные для решения задач на сложных типах данных.

При таком подходе реальные алгебраические машины, работающие с конкретными типами данных, разрабатываются как АТД – наследники абстрактных алгебраических машин, у которых они наследуют операции над структурами и включают в себя операции над реальными элементами этих структур.

Одно из основных понятий некоторых, например, широко используемой реляционной модели, моделей данных – понятие кортежа. Поэтому проблема построения АТД, соответствующих реальным алгебраическим машинам, на основе которых строятся такие распространенные модели данных как реляцион-

ная, многомерная, NoSQL, неотделима от проблемы создания алгебры кортежей произвольной структуры. Далее рассматривается способ построения бинарных операций над кортежами, без которых невозможно построение бинарных операций над агрегатами данных, которые используются в качестве элементов структуры (отношениями, файлами, многомерными матрицами). В различных моделях данных эти операции над кортежами, заданные явно или неявно, интерпретируются либо как аддитивные, либо как мультипликативные. Предложенный метод [117] позволяет делать явные формальные описания бинарных операций над кортежами. Далее используется общепринятое в математике определение кортежа.

Определение 2.12. Кортеж – это конечный набор (t_1, …, t_n) длины n, (где n – неотрицательное целое число), каждый элемент которого t_i принадлежит некоторому типу T_i ($1 \le i \le n$).

Важную роль в моделях данных играют нуль-кортежи. В дальнейшем нуль-кортеж рассматривается как кортеж, состоящий только из нейтральных элементов типов T_1, …, T_n,.

Предполагается, без ограничения общности, что кортежи не могут содержать сложные элементы, например, другие кортежи.

Пусть T_1,…, T_n – совокупность основных множеств универсальных алгебр или алгебраических систем, которые принято в языках программирования называть простыми типами. Это, например, числа с фиксированной или плавающей точкой, строки, а также типы, полученные из простых типов добавлением новых операций. Пусть x_1, …, x_p, y_1,…, y_q, ($0 \le p, q \le n, p+q=n$)–набор переменных, каждая из которых принимает значения в одном и только одном из множеств T_1,…, T_m. На этих множествах определяется система функций: $f^{j}_{\alpha_1..\alpha_k,\beta_1..\beta_l}(x_{\alpha_1},...,x_{\alpha_k},y_{\beta_1},...,y_{\beta_l}):(T_{\alpha_1}\times...\times T_{\alpha_k}\times T_{\beta_1}\times...\times T_{\beta_l})\to T_i$. Здесь выполняются неравенства $1 \le k \le p$, $1 \le l \le q$, $j > 0$ и $1 \le i \le n$. Далее будет использоваться сокращенная запись этих функций – $f^{j}_{\alpha_1..\alpha_k,\beta_1..\beta_l}$.

Определение 2.13. Пусть кортеж c_1 длины p и кортеж c_2 длины q составлены из переменных $x_1, \ldots, x_p$ и $y_1,\ldots, y_q$ соответственно. Тогда кортеж c_3 длины r, построенный по правилу $(f^1_{\alpha_1..\alpha_k,\beta_1...\beta_l}, \ldots, f^r_{\alpha_1..\alpha_k,\beta_1...\beta_l})$, можно рассматривать как результат бинарной операции над кортежами c_1 и c_2 ($c_3 = c_1 * c_2$). Если функция определена на всех элементах кортежей c_1 и c_2, то используется обозначение $f^j_{c_1,c_2}$.

Семантика операции над кортежами (аддитивность или мультипликативность) определяется семантикой операции, определенной над структурой, типами элементов которой могут быть кортежи c_1, c_2 и c_3.

В следующем примере показано построение бинарных операций над кортежами.

Пример 2.6. Пусть S – множество строк, а R^+ –множество положительных действительных чисел. Переменные (атрибуты) A, B, C, D принимают значения в множестве S, а переменные X, Y – в множестве R^+. Кортежи c_1 и c_2 имеют схемы $c_1(A, B, C, X)$ и $c_2(B, C, D, Y)$. Функции $f^2_{c_1,c_2}(d) = d, (d \in D)$, позволяют построить кортеж $c_3 = c_1 \times c_2$ со схемой $c_3(A, D, Z)$, где $Z = X \times Y$, и принимает значения в множестве R^+. Таким образом определяется мультипликативная операция над кортежами. Если c_{31}, c_{32}, c_{33} – кортежи со схемой $c_3(A, D, Z)$, то функции

$$f^1_{c_{31},c_{32}}(c_{31}.a) = c_{31}.a, (a \in A),$$

$$f^2_{c_{31},c_{32}}(c_{31}.d) = c_{31}.d, (d \in D),$$

$$f^3_{c_{31},c_{32}}(c_{31}.z, c_{32}.z) = c_{31}.z \times c_{32}.z, (z \in Z)$$

определяют аддитивную операцию над кортежами $c_{33} = c_{31} + c_{32}$.

Если предположить, что в реляционной модели кортежи c_1 и c_2 есть элементы (строки) отношений R_1 и R_2, то рассмотренные функции позволяют сформировать список полей в выражении, реализующем запрос:

```
SELECT R1.A, R2.D, Sum(R1.X*R2.Y) AS Z
FROM R1 INNER JOIN R2 ON (R1.C = R2.C) AND (R1.B = R2.B)
GROUP BY R1.A, R2.D;
```

Подобным образом рассмотренные функции могут быть использованы для построения алгебр элементов структур (файлов и многомерных матриц) в теоретико-множественной и многомерно-матричной моделях.

В некоторых моделях при построении бинарных операций над элементами структуры может возникнуть ситуация, когда кортеж-результат формируется только из одного кортежа-операнда. В многомерно-матричной модели такое невозможно, так как в матрице присутствуют все возможные элементы, в том числе и нуль-кортежи. Но в реляционной и теоретико-множественной моделях отношения и файлы, как правило, не содержат строки и записи, состоящие только из нейтральных элементов. Поэтому целесообразно рассмотреть еще два вида функций $f^{j}_{\alpha_1..\alpha_k,0}(x_{\alpha_1},...,x_{\alpha_k}):(T_{\alpha_1}\times...\times T_{\alpha_k})\to T_j$, и $f^{j}_{0,\beta_1...\beta_l}(y_{\beta_1},...,y_{\beta_l}):(T_{\beta_1}\times...\times T_{\beta_l})\to T_j$, которые позволят конструировать операции, формирующие кортеж-результат только из одного кортежа-операнда. Использование этих операций обеспечивает единство формальной записи алгоритмов бинарных операций над структурами. Сокращенная запись этих функций имеет вид: $f^{j}_{0,\beta_1...\beta_l}$, $f^{j}_{\alpha_1..\alpha_k,0}$.

Предложенный способ позволяет определять различные аддитивные и мультипликативные операции над кортежами и получать при этом различные универсальные алгебры или алгебраические системы. Свойства построенных операций составляют список аксиом этих алгебраических систем. Благодаря этому возникает возможность формального построения произвольных универсальных алгебраических систем кортежей в соответствии с требованиями реальных задач.

От построенной таким образом универсальной алгебраической системы легко перейти к представлению АТД *Кортеж*. Такой АТД является базовым объектом, который может стать родоначальником множества объектов, соответствующих реальным задачам. Эти объекты будут, с одной стороны, наследовать свойства составляющих кортежи простых типов, с другой, базовые операции объекта, представляющего АТД *Кортеж*. Таким образом, в объектах-

наследниках могут использоваться либо основные операции над кортежами из базового объекта, либо заменяющие их полиморфные операции, которые будут выполнять те действия над элементами простых типов данных, которые требуются в условиях конкретной решаемой задачи.

Способ построения универсальной алгебраической системы для кортежей будет использован в дальнейшем для построения второго основного множества (типа) в абстрактных файловой и матричной алгебраических машинах. Он также будет использован для доказательства изоморфизма рассматриваемых в работе моделей данных.

2.2. Файловая (теоретико-множественная) модель данных

Как было отмечено в разделе 1.1 СУБД опирается на файловую систему, присущую конкретному вычислительному комплексу или операционной системе. Поэтому, по крайней мере, одна из промежуточных моделей данных должна обеспечивать формализм, необходимый для решения задач распараллеливания обработки.

2.2.1. Анализ определений файла

Для построения файловой модели данных необходимо понять, что представляет собой этот агрегат данных. Для этого проводится анализ известных определений файла.

В операционных системах понятие файла связано исключительно с хранением данных во внешней памяти. Поэтому самое общее определение файла таково: "с точки зрения прикладной программы файл – это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные" [72].

Однако с позиции систем обработки данных, файл – это поименованная совокупность данных, содержащих достаточно полное описание некоторой материальной совокупности, состоящей из однородных или однотипных объектов. Файл обычно принято подразделять на записи, каждая из которых описы-

вает, как правило, один из таких объектов [131]. Эта точка зрения разделяется и в следующих определениях:

1. Файл – поименованная совокупность всех экземпляров логических записей заданного типа [132].
2. Файл – именованная структура данных, представляющая собой множество записей какого-либо типа, хранимых во внешней памяти. В некоторых системах так называют структуру хранимых данных [70].

В этих определениях существенное значение имеет понятие логической записи и ее экземпляра. В дальнейшем, поскольку файл будет определяться как множество, под логической записью будет пониматься описание элементов этого множества, а под экземплярами – конкретные элементы.

Для конструирования логических записей используется понятие поля, или хранимого поля. В таком контексте файл определяется следующим образом [133].

Хранимое поле – это наименьшая единица хранимых данных. Как правило, хранимые поля – это переменные, значения которых берутся из простых типов данных: чисел, строк и тому подобных. Типичная база данных содержит множество экземпляров каждого из нескольких описанных в ней типов хранимых полей.

Хранимая запись – это набор взаимосвязанных хранимых полей, а экземпляр хранимой записи состоит из группы связанных экземпляров хранимых полей.

Хранимый файл – это набор всех существующих в настоящий момент экземпляров хранимых записей одного и того же типа. Для упрощения предполагается, что любой заданный хранимый файл может содержать хранимые записи только одного типа.

В общем виде, с использованием формы Бэкуса-Наура (БНФ) так как это делается в [134], файл можно определить следующим образом.

<файл> ::= <запись-метка> <КФ> | <запись-метка> <строка записей><КФ>

<строка записей> ::=<запись> | <строка записей><запись>

<запись>::={<элемент>}

<запись-метка> ::= идентификатор файла

<КФ> ::= признак конца файла

<элемент>::= переменная простого или структурного типа.

Из этого определения следует, что файл может быть либо пустым, не содержащим ни одной записи, либо содержать последовательность (строку) записей. Записи состоят из элементов, каждый из которых есть переменная либо простого типа, либо структурного типа при условии, что на этом структурном типе определены необходимые для решения конкретной задачи бинарные операции. То есть, элемент записи не может иметь тип массив, но может иметь тип вектор или тип матрица.

Все рассмотренные определения дают достаточно точное описание файла и составляющих его записей. Однако для описания операций над файлами, которые используются в системах обработки данных, этих определений недостаточно. Поэтому далее приводится строгая алгебраическая модель файловой обработки данных. При создании этой модели [73, 74] были использованы методы, предложенные в [61, 62].

2.2.2. Определение файла

Формализация понятия файла начинается с формализации понятия запись. Определение понятия запись в предыдущем разделе позволяет в дальнейшем рассматривать записи как структурные типы данных, объединяющие под одним именем разнотипные переменные.

Пусть $A=\{A_1, \ldots, A_p\}$ – некоторая конечная система конечных множеств, а $N=\{N_1, \ldots, N_p\}$ – конечное множество элементов, называемых *именами* множеств $A_1, \ldots, A_p$. Множества $A_1, \ldots, A_p$ могут состоять из элементов любой природы: чисел с фиксированной или плавающей точкой, строк, а также таких структур как векторы, матрицы, кортежи, с заданными на них операциями и тому подобных. Так как на множествах $A_1, \ldots, A_p$ заданы операции и отноше-

ния, то A_1, …, A_p в дальнейшем будут рассматриваться как простые или абстрактные типы данных.

Определение 2.14. Полем записи называется пара $F=<N_i, A_i>$ (i=1, …, p), где N_i – имя, а A_i – множество значений поля.

Определение 2.15. Кортеж $R=\{F_1, \ldots, F_p\}$ называется записью типа R.

Определение 2.16. Кортеж вида $R^* = \{< N_1, A_1^* >, ..., < N_p, A_p^* >\}$, где $(A_i^* \in A_i, i = 1, ..., p)$ называется экземпляром записи типа R.

В дальнейшем изложении термин запись будет отождествляться с термином экземпляр записи типа R в тех случаях, когда это не будет вызывать неоднозначного толкования. При необходимости и из соображений удобства записи будут отождествляться с синтаксическими конструкциями *struct* и *record*, используемыми в языках программирования для задания записей, а экземпляры записей – со строками таблиц.

Определение 2.17. Множество $\boldsymbol{X}$ экземпляров записей типа R называется множеством записей типа R или множеством однотипных записей.

Для определения операций над файлами необходимо выделить некоторые структурные элементы записей, которые позволят идентифицировать записи и, при необходимости, сравнивать их. Во всех моделях данных такие элементы называются ключами.

Определение 2.18. Пусть $K = \{K_1, ..., K_m\}, (m < p)$ – множество полей записи R, такое, что $K_1 = F_{\alpha_1}, ..., K_m = F_{\alpha_m}$, причем все A_{α_i} $(i = 1, ..., m)$ – типы, на которых заданы отношения эквивалентности и порядка. Множество K называется множеством ключей, а его элементы ключами.

Определение 2.19. Кортеж $K^* = \{K_1^*, ..., K_m^*\}$, для элементов которого выполняется правило $K_i^* \in A_{\alpha_i}$ $(i = 1, ..., m)$, называется экземпляром множества ключей (K_i^* называется экземпляром ключа).

Очевидно, что любая совокупность экземпляров множества ключей может быть лексикографически упорядочена. Одновременно упорядочивается и множество однотипных записей $\boldsymbol{X}$, тип которых включает множество ключей K.

Определение 2.20. Две однотипные записи называются эквивалентными, если они содержат одинаковые экземпляры множества ключей.

В современных языках программирования существует возможность создавать под одним и тем же именем записи с различной структурой. Это не нарушает однотипность записей в рассмотренном смысле при условии, что каждый вариант структуры записи содержит все ключи из множества K.

Таким образом, задание на множестве однотипных записей $\boldsymbol{X}$ множества ключей K разбивает $\boldsymbol{X}$ (индуцирует разбиение $\boldsymbol{X}$) на группы (классы), содержащие записи с одинаковыми значениями ключей – эквивалентные записи. Эти классы называются *классами эквивалентности*. Они могут содержать разное количество записей.

Совокупность всех классов эквивалентности по отношению, заданному множеством ключей, образует *фактор-множество* множества однотипных записей $\boldsymbol{X}$.

В дальнейшем такое фактор-множество будет обозначаться X_K, составляющие его классы эквивалентности – X_{K^*}, или $X_{K^*_{(1)}}, X_{K^*_{(2)}}, \ldots$. Также будет полезно рассматривать экземпляры множества ключей, которым во множестве $\boldsymbol{X}$ не соответствует ни одной записи. В этом случае, при определении операций над файлами, целесообразно считать, что таким экземплярам множества ключей соответствует универсальная неопределенная запись Θ. Класс эквивалентности, соответствующий экземпляру множества ключей K^* и состоящий из единственной записи Θ, будет обозначаться Θ_{K^*}

Определение 2.21. Пусть даны множество однотипных записей X и множество ключей K. Файлом X_K называется фактор-множество множества однотипных записей X по отношению эквивалентности, порожденному множеством K.

Как и фактор-множество, файл обозначается X_K, а класс эквивалентности, соответствующий экземпляру множества ключей K^*, обозначается X_{K^*}.

При таком подходе файл не может быть неупорядоченным. Это, в некоторой степени, не соответствует общепринятому представлению о файлах, как носителях произвольной информации. Но в дальнейшем рассматриваются только такие файлы, которые несут в себе структурированную информацию об объектах предметных областей и как носители этой информации используются на входе, выходе и в процессе решения задач, поставленных и формализованных на этих предметных областях. Процессы решения таких задач построены на основе набора операций обработки файлов, которые реализуются алгоритмами, весьма чувствительными к упорядоченности входных файлов. Это особенно заметно при обработке больших объемов данных.

Определение 2.22. Если каждый класс эквивалентности файла X_K содержит единственную запись, то файл X_K называется строго упорядоченным, если же в каждом классе эквивалентности может быть более одной записи – нестрого упорядоченным.

Это определение завершает построение теоретико-множественной модели файла – носителя информации об объектах предметных областей. В терминах этой модели легко задать теоретико-множественные описания операций над файлами.

2.2.3. Описание операций над файлами

Поскольку операции над файлами определяются с учетом выполнения операций над записями, необходимо выбрать модель операции над кортежами, которым, согласно определению, соответствуют записи файлов. Очевидно, что запись можно рассматривать как пару кортежей $R=\{K, D\}$, где кортеж $K=\{K_1, ..., K_m\}, (m<p)-$ множество ключей, а кортеж $D=\{F_{m+1}, ..., F_p\}$ – множество неключевых полей, участвующих в вычислениях, но не участвующих в идентификации и сравнении записей. Записи, в которых кортеж D – нуль-кортеж, физически в файле не присутствуют. Логически таким экземплярам

множества ключей *K* соответствуют универсальные неопределенные записи Θ и классы эквивалентности Θ_{K^*}. Поэтому для построения бинарных операций над записями выбрана следующая модель бинарной операции над кортежами:

$$c_1(x_1,...,x_p)*c_2(y_1,...,y_q)=\begin{cases} c_3(f^{\gamma_1}_{\alpha_1..\alpha_{k_1},0},...,f^{\gamma_n}_{\alpha_1..\alpha_{k_n},0}),\ q=0;\\ c_3(f^{\gamma_1}_{0,\beta_1..\beta_{l_1}},...,f^{\gamma_n}_{0,\beta_1..\beta_{l_n}}),\ p=0;\\ c_3(f^{\gamma_1}_{\alpha_1..\alpha_{k_1},\beta_1..\beta_{l_1}},...,f^{\gamma_n}_{\alpha_1..\alpha_{k_n},\beta_1..\beta_{l_n}}),\ p\neq 0,\ q\neq 0. \end{cases}$$

Исходя из этого, будет построена следующая система операций над файлами.

Сортировка (*sort*). Выполнение операции сортировки приводит к построению из исходного множества однотипных записей *X* файла X_K (фактор-множества *X* по заданному множеству ключей *K*). Практические соображения требуют, чтобы для любого множества *X* можно было подобрать такое множество ключей *K*, по которому файл X_K будет строго упорядоченным.

Выборка (*sel*). Пусть даны файл X_K и $\pi(K)$ предикат, определенный на множестве ключей *K*. Операция выборки приводит к созданию файла X^{π}_{K}, удовлетворяющего следующим условиям:

$X^{\pi}_{K}\subseteq X_K$, то есть файл X^{π}_{K} есть подмножество файла X_K;

$\forall K^*(X_{K^*}\in X^{\pi}_{K}\wedge\pi(K^*))$, то есть класс эквивалентности X_{K^*} присутствует в файле X^{π}_{K} тогда и только тогда, когда все значения ключей в экземпляре множества ключей *K** превращают предикат $\pi(K)$ в истинное высказывание.

Таким образом, в результате выполнения операции выборки образуется подмножество исходного файла. Принадлежащие этому подмножеству записи содержат фиксированные экземпляры одного или нескольких ключей. Если фиксируются значения не всех ключей, то остальные ключи принимают все допустимые значения.

Сжатие (*quant*). Пусть даны файлы X_K, нестрого упорядоченный по множеству ключей *K*, и Y_K, строго упорядоченный по множеству ключей *K*. Классы эквивалентности этих файлов связаны соотношением $Y_{K^*}=f(X_{K^*})$, где $f-$ функ-

ция, реализующая групповую операцию (операцию квантификации). Тогда считается, что файл Y_K получен из файла X_K в результате применения операции сжатия.

В операциях сжатия каждому непустому классу эквивалентности X_{K^*} исходного файла, нестрого упорядоченного по множеству ключей *K*, ставится в соответствие класс Y_{K^*} выходного файла Y_K, строго упорядоченного по тому же множеству ключей. При этом единственная запись, принадлежащая Y_{K^*}, вычисляется как значение функции *f*, определенной на всех записях класса эквивалентности X_{K^*}. В качестве такой функции могут быть использованы, например, такие агрегатные функции как вычисление суммы или среднего значения, поиск наименьшего или наибольшего значения и тому подобные.

Слияние строго упорядоченных файлов (*ms*). Пусть даны два файла X_K и Y_K строго упорядоченные по одному и тому же множеству ключей *K*. В результате слияния этих строго упорядоченных файлов образуется файл Z_K, классы эквивалентности задаются соотношением $Z_{K^*} = f(X_{K^*}, Y_{K^*})$. Функция $f(X_{K^*}, Y_{K^*})$, определенная на классах эквивалентности исходных файлов, задает характер операции. Следующие примеры демонстрируют построение этой функции в зависимости от решаемой задачи.

Пример 2.7.Если слияние строго упорядоченных файлов используется для реализации теоретико-множественной операции объединения файлов, то функция *f* задается как: $f(X_{K^*}, Y_{K^*}) = \begin{cases} Y_{K^*}, \textit{если } X_{K^*} = \Theta_{K^*}, \\ X_{K^*}, \textit{в противном случае} \end{cases}$, для операции пересечения: $f(X_{K^*}, Y_{K^*}) = \begin{cases} Y_{K^*}, \textit{если } X_{K^*} \neq \Theta_{K^*} \textit{ и } Y_{K^*} \neq \Theta_{K^*}, \\ \Theta_{K^*}, \textit{в противном случае} \end{cases}$.

Для реализации теоретико-множественной операции симметрической разности файлов задание функции *f* будет таким:

$$f(X_{K^*},Y_{K^*})=\begin{cases}Y_{K^*},\ если\ X_{K^*}=\Theta_{K^*}\\ X_{K^*},\ если\ Y_{K^*}=\Theta_{K^*}\\ \Theta_{K^*},\ если\ X_{K^*}\neq\Theta_{K^*},Y_{K^*}\neq\Theta_{K^*}\end{cases}.$$

Пример 2.8. Распространенная операция корректировки одного файла при помощи другого задается следующим образом:

$$f(X_{K^*},Y_{K^*})=\begin{cases}X_{K^*},\ если\ Y_{K^*}=\Theta_{K^*}\\ Y_{K^*},\ если\ X_{K^*}=\Theta_{K^*}\\ Y_{K^*},\ если\ (X_{K^*}\neq\Theta_{K^*}\wedge Y_{K^*}\neq\Theta_{K^*}\wedge\pi(Y_{K^*}))\\ \Theta_{K^*},\ если\ X_{K^*}\neq\Theta_{K^*}\wedge Y_{K^*}\neq\Theta_{K^*}\wedge\neg\pi(Y_{K^*}))\end{cases}$$

Предикат $\pi(Y_{K^*})$ служит признаком замены или удаления записи корректируемого файла.

Пример 2.9. В сложных вычислительных задачах построение функции f может быть следующим:

$$f(X_{K^*},Y_{K^*})=\begin{cases}g_1(Y_{K^*}),\ если\ X_{K^*}=\Theta_{K^*}\\ g_2(X_{K^*}),\ если\ Y_{K^*}=\Theta_{K^*}\\ g_3(X_{K^*},Y_{K^*}),\ если\ X_{K^*}\neq\Theta_{K^*},Y_{K^*}\neq\Theta_{K^*}\end{cases}$$

Эта формула обеспечивает формирование записей выходного файла Z_K по следующим правилам:

– одноместные функции g_1 и g_2 определены на записях файлов X_K и Y_K, соответственно, и реализуются при помощи функций $f^{j}_{0,\beta_1\ldots\beta_l}$, $f^{j}_{\alpha_1\ldots\alpha_k,0}$;

– двухместная функция g_3 определена на парах записей, принадлежащих декартову произведению $X_K\times Y_K$, и реализуется при помощи функций $f^{j}_{\alpha_1\ldots\alpha_k,\beta_1\ldots\beta_l}$;

– функции g_1, g_2 и g_3 обеспечивают формирование записи выходного файла с вычислением новых значений неключевых полей, из значений неключевых полей записей X_{K^*} и Y_{K^*};

– значения ключей все три функции переносят в выходную запись Z_{K^*} без изменений.

Слияние нестрого упорядоченных файлов (*mns*). Пусть X_L и Y_M – файлы, упорядоченные (возможно строго) по множествам ключей L и M, причем выполняется условие $L \cap M \neq \varnothing$, и пусть K – множество ключей, связанное с множествами L и M соотношениями:

1. $K \subseteq L \cup M$,
2. $L \cap K \neq \varnothing$ и $M \cap K \neq \varnothing$.

Это означает, что множество ключей K состоит из ключей, входящих в множества L и M, причем в K содержится, по крайней мере, по одному ключу из каждого множества. Тогда, по крайней мере, один файл $X_{K \cap L}$ или $Y_{K \cap M}$ нестрого упорядочен по множеству ключей. Если $M \not\subset L$ и $L \not\subset M$, то файлы $X_{K \cap L}$ и $Y_{K \cap M}$ нестрого упорядочены. Слияние файлов производится по множеству ключей K. Пусть K^* – фиксированный экземпляр множества ключей K, а $(K \cap L)^*$ и $(K \cap M)^*$ – такие фиксированные экземпляры множеств ключей $(K \cap L)$ и $(K \cap M)$, что значения одноименных ключей в них совпадают. Тогда можно задать вычисление класса эквивалентности файла Z_K по следующему правилу:

$$Z_{K^*} = \begin{cases} \Theta_{K^*}, & \text{если } X_{(K \cap L)^*} = \Theta_{(K \cap L)^*}, \text{ или } Y_{(K \cap M)^*} = \Theta_{(K \cap M)^*}, \\ f(X_{(K \cap L)^*}, Y_{(K \cap M)^*}), & \text{в противном случае.} \end{cases}$$

Функция $f(X_{(K \cap L)^*}, Y_{(K \cap M)^*})$ определена на классах эквивалентности $X_{(K \cap L)^*}$ и $Y_{(K \cap M)^*}$, а ее значение – класс эквивалентности Z_{K^*}, состоящий из элементов, каждый из которых вычисляется из пары элементов, принадлежащей декартову произведению $X_{(K \cap L)^*} \times Y_{(K \cap M)^*}$.

Если функция $f(X_{(K \cap L)^*}, Y_{(K \cap M)^*})$ реализует групповую операцию, то операция слияния нестрого упорядоченных файлов включает в себя и операцию сжатия файла Z_K, в результате которой получается файл $Z_{K'}$ (множество ключей K' есть подмножество множества ключей K, то есть $K' \subset K$).

Определенные таким образом операции над файлами позволяют построить двухосновную алгебраическую систему или АТД, называемый абстрактной алгебраической файл-машиной.

Пусть $\mathscr{F}$ – множество всех возможных файлов. Тогда предложенные унарные и бинарные операции можно рассматривать как отображения $\mathscr{F} \to \mathscr{F}$ и $\mathscr{F} \times \mathscr{F} \to \mathscr{F}$. Кроме того, для работы с файлами необходим предикат $eof : \mathscr{F} \to \{0, 1\}$.

Если $\mathscr{R}$ – множество всех записей (всех возможных типов), то операции $get : \mathscr{F} \to \mathscr{R}$ и $put : \mathscr{F} \times \mathscr{R} \to \mathscr{F}$ – имеющиеся во всех языках программирования, операции чтения и записи файла. Кроме того, в задачах обработки файлов определяются аддитивная и мультипликативная операции $+, \times : \mathscr{R} \times \mathscr{R} \to \mathscr{R}$ и предикаты $pred, eq, succ : \mathscr{R} \times \mathscr{R} \to \{0, 1\}$, которые задают на множестве записей бинарные отношения *предшествования*, *эквивалентности* и *следования за*.

Таким образом, построена абстрактная алгебраическая файл-машина

$F_{am} = <\mathscr{F}, \mathscr{R}; sort, sel, quant, ms, mns, get, put, +, \times; eof, pred, eq, succ>$.

Реализация такого абстрактного типа данных достаточно проста во всех современных языках программирования.

2.3. Многомерно-матричная модель данных

2.3.1. Задачи многомерно-матричного представления данных

Рассматриваемая далее многомерно-матричная модель данных была предложена в [73, 74] с целью решения задач повышения эффективности обработки файлов.

В этом состоит ее существенное отличие от модели, предложенной в [135-136], которая была ориентирована на пользователя-аналитика, в предположении, что "он видит мир предприятия многомерным по своей природе". Поскольку, предполагалось, что пользователь должен выполнять аналитическую обработку в реальном времени, эта модель получила название OLAP (online analytical processing). По мнению автора, Э. Ф. Кодда, из многомерных пред-

ставлений пользователя следует, что "OLAP-модель должна быть многомерной в своей основе. Многомерная концептуальная схема или пользовательское представление облегчают моделирование и анализ так же, впрочем, как и вычисления". На основе OLAP-модели была разработана OLAP-технология обработки данных, заключающаяся в подготовке суммарной (агрегированной) информации на основе больших массивов данных, структурированных по многомерному принципу. OLAP-технология реализована во многих современных прикладных системах, от самых простых – процессоров электронных таблиц, подобных MS Excel, до больших и высокопроизводительных СУБД, таких как MSSQL-server и Oracle. Для реализации вычислений разработаны специальные языки, такие как MDX [137]. Упрощенно, решение задачи в OLAP-технологии состоит из двух шагов: создания многомерной структуры, которая называется гиперкубом, OLAP-кубом или просто кубом, и выполнения унарных операций, реализующих запросы к этой структуре. Поскольку объемы обрабатываемых данных настолько велики, что этот класс задач можно отнести к направлению Big Data, возникает потребность в оптимизации хранения и обработки данных. Часто на практике оптимизация одного шага приводит к ухудшению характеристик другого. Это происходит потому, что:

1. затруднительно эффективно построить многомерную структуру даже из данных, размещенных в аналогичных структурах, если не использовать специальных алгебраических операций над этими структурами;
2. большинство современных методов оптимизации процессов обработки данных в различных моделях данных основаны на эвристическом подходе, что не позволяет эффективно использовать специфические свойства многомерных структур данных и операций над ними [65, 138].

Решение этих проблем возможно на основе использования алгебры многомерных матриц [58], свойства которой позволяют решать перечисленные проблемы. Операции алгебры многомерных матриц достаточно просто реализуются на параллельных вычислительных комплексах с различными архитектурами. Одно из важнейших свойств алгебры многомерных матриц – возможность по-

строения оптимальных процессов обработки данных на основе использования формальных методов оптимизации, например, динамического программирования. Поэтому в работе алгебра многомерных матриц используется для решения задач повышения эффективности процессов обработки данных на современных вычислительных комплексах.

2.3.2. Алгебра многомерных матриц

Обычно, под матрицами понимают структурированные совокупности элементов простых типов. Далее рассматриваются многомерные матрицы, элементы которых могут принадлежать произвольным типам данных. Главное и единственное требование состоит в том, что на этих типах должны быть определены две алгебраические операции: аддитивная и мультипликативная. Это означает, что типы элементов матриц должны быть, по крайней мере, группоидами по каждой из определенных на них операции.

Такой подход позволяет использовать для конструирования матриц, в том числе и такие стандартные структурные типы как векторы и матрицы, хотя использование многомерных матриц лишает подобные конструкции практического смысла. В принципе, в качестве типов элементов матриц могут быть использованы различные АТД. Для достижения поставленных в работе целей особенно важно, что для построения многомерно-матричной модели данных могут быть использованы сконструированные в соответствии с указанным требованием АТД для различных типов кортежей.

Определение 2.23. Пусть $i_1,...,i_p$ – совокупность индексов, принимающих значения от 1 до $n_\alpha (\alpha = 1, ..., p)$ соответственно. Тогда p-мерная матрица – это совокупность $T = \{a_{i_1 ... i_p}\}$ элементов некоторого типа, над которым определены аддитивная и мультипликативная операции.

Таким образом, p-мерная матрица содержит $n_1 \times ... \times n_p$ элементов. Для многомерных матриц используется обозначение $A = \left\| a_{i_1 ... i_p} \right\|$. Сигнатура алгебры многомерных матриц содержит унарные операции транспонирования, сечения,

свертки и бинарные операции сложения и умножения. Далее приводятся определения этих операций.

Транспонирование. Матрица $A' = \left\| a_{i_{\alpha_1} \ldots i_{\alpha_p}} \right\|$, элементы которой связаны с элементами матрицы $A = \left\| a_{i_1 \ldots i_p} \right\|$ соотношением $a_{i_{\alpha_1} \ldots i_{\alpha_p}} = a_{i_1 \ldots i_p}$, где $(i_{\alpha_1}, ..., i_{\alpha_p})$ – какая-нибудь перестановка индексов $(i_1, ..., i_p)$, называется транспонированной относительно матрицы *A* соответственно этой перестановке.

Визуально *p*-мерная матрица может быть представлена в виде *p*-мерного параллелепипеда или гиперкуба. Следовательно, операцию транспонирования можно интерпретировать как вращение этого параллелепипеда или гиперкуба.

Сечение. Возможны два варианта этой операции. В первом размерность матрицы уменьшается, а во втором остается прежней.

Простое *m*-кратное сечение. Пусть в *m* индексах ($1 \le m \le p$) совокупности индексов $(i_1, ..., i_p)$ матрицы зафиксировано по одному значению. Для простоты и без ограничения общности можно считать, что это индексы $(i_1, ..., i_m)$. $(p-m)$-мерная матрица $\left\| A_{(i_1^0, \ldots, i_m^0, i_{m+1}, \ldots, i_p)} \right\|$, состоящая только из тех элементов матрицы $A = \left\| a_{i_1 \ldots i_p} \right\|$, в которых индексы $(i_1, ..., i_m)$ имеют единственное фиксированное значение $(i_1^0, ..., i_m^0)$, называется простым *m*-кратным сечением матрицы *A* ориентации $(i_1, ..., i_m)$.

Пример 2.9. Пусть $A = \left\| a_{i_1 i_2 i_3 i_4} \right\|$ – четырехмерная матрица, все индексы которой принимают значения 1, 2. Если зафиксировать значения двух индексов i_1=2 и i_2=1, то получается двукратное простое сечение матрицы *A* ориентации (i_1, i_2), которое представляет собой двумерную матрицу вида: $A_{(i_1, i_2)} = \left\| \begin{matrix} a_{2111} & a_{2112} \\ a_{2121} & a_{2122} \end{matrix} \right\| \begin{pmatrix} i_1 = 2 \\ i_2 = 1 \end{pmatrix}$.

Если зафиксировать значение одного индекса i_2=2, то получается однократное простое сечение матрицы A ориентации (i_2), которое представляет собой трехмерную матрицу вида: $A_{(i_2)} = \begin{array}{c} \\ i_1, i_3 \downarrow \end{array} \begin{array}{c} i_4 \rightarrow \\ \left\| \begin{array}{cc} a_{1211} & a_{1212} \\ a_{1221} & a_{1222} \\ \hline a_{2211} & a_{2212} \\ a_{2221} & a_{2222} \end{array} \right\| \end{array} (i_2 = 2)$.

Сечение с фиксированными значениями индексов. Пусть в множествах значений индексов $(i_1, ..., i_m)$ (1≤m≤p) совокупности индексов $(i_1, ..., i_p)$ матрицы $A = \left\| a_{i_1 \ldots i_p} \right\|$ зафиксировано более чем по одному значению. Это означает, что любой индекс i_k (1≤k ≤m) принимает t_k (1<t_k<n_k) значений из множества (1, …, n_k). p-мерная матрица $A_{(i_1, ..., i_m)}$, состоящая только из тех элементов матрицы $A = \left\| a_{i_1 \ldots i_p} \right\|$, в которых индексы $(i_1, ..., i_m)$ принимают соответственно t_1, …, t_k значений, называется m-кратным сечением ориентации $(i_1, ..., i_m)$ с фиксированными значениями индексов матрицы A.

Очевидно, что m-кратное сечение матрицы A можно рассматривать как матрицу, построенную из простых m-кратных сечений.

Пример 2.10. Пусть $A = \left\| a_{i_1 i_2 i_3 i_4} \right\|$ – четырехмерная матрица, у которой все индексы принимают значения (1, 2, 3). С помощью двукратных сечений эту матрицу можно представить в виде:

$$A = \begin{array}{c} \\ i_1 \downarrow \end{array} \begin{array}{c} i_2 \rightarrow \\ \left\| \begin{array}{ccc|ccc|ccc} a_{1111} & a_{1112} & a_{1113} & a_{1211} & a_{1212} & a_{1213} & a_{1311} & a_{1312} & a_{1313} \\ a_{1121} & a_{1122} & a_{1123} & a_{1221} & a_{1222} & a_{1223} & a_{1321} & a_{1322} & a_{1323} \\ a_{1131} & a_{1132} & a_{1133} & a_{1231} & a_{1232} & a_{1233} & a_{1331} & a_{1332} & a_{1333} \\ \hline a_{2111} & a_{2112} & a_{2113} & a_{2211} & a_{2212} & a_{2213} & a_{2311} & a_{2312} & a_{2313} \\ a_{2121} & a_{2122} & a_{2123} & a_{2221} & a_{2222} & a_{2223} & a_{2321} & a_{2322} & a_{2323} \\ a_{2131} & a_{2132} & a_{1133} & a_{2231} & a_{2232} & a_{2233} & a_{2331} & a_{2332} & a_{2333} \\ \hline a_{3111} & a_{3112} & a_{3113} & a_{3211} & a_{3212} & a_{3213} & a_{3311} & a_{3312} & a_{3313} \\ a_{3121} & a_{3122} & a_{3123} & a_{3221} & a_{3222} & a_{3223} & a_{3321} & a_{3322} & a_{3323} \\ a_{3131} & a_{3132} & a_{3133} & a_{3231} & a_{3232} & a_{3233} & a_{3331} & a_{3332} & a_{3333} \end{array} \right\| \end{array}.$$

Матрица *A* представляет собой четырехмерный гиперкуб. Если зафиксировать значения индексов i_1=(1, 2) и i_2=(1, 3) то в результате выполнения операции сечения матрица *A* преобразуется в четырехмерный параллелепипед вида:

$$A_{(i_1,i_2)} = \begin{array}{c} \\ i_1 \downarrow \end{array} \overset{\begin{array}{c} \\ i_2 \rightarrow \end{array}}{\left\| \begin{array}{cc|cc} a_{1111} & a_{1113} & a_{1311} & a_{1313} \\ a_{1113} & a_{1123} & a_{1321} & a_{1323} \\ \hline a_{1131} & a_{1133} & a_{1331} & a_{1333} \\ \hline a_{2111} & a_{2113} & a_{2311} & a_{2313} \\ a_{2121} & a_{2123} & a_{2321} & a_{2323} \\ a_{2131} & a_{1133} & a_{2331} & a_{2333} \end{array} \right\|} \begin{pmatrix} i_1 = (1,2) \\ i_1 = (2,3) \end{pmatrix}.$$

Возможен и комбинированный вариант этой операции, когда по части индексов выполняется простое сечение, а по части – сечение с фиксированным набором значений индексов. В этом случае размерность матрицы-результата меньше размерности матрицы-операнда на число индексов, по которым производится простое сечение.

Свертка. Пусть дано разбиение совокупности индексов матрицы $A = \left\| a_{i_1 \ldots i_p} \right\|$ на совокупности *l*=(l_1,…,l_κ) и *c*=(c_1,…, c_μ), κ+μ= *p*. Матрица ${}^{\mu}A = \left\| a_l \right\|$, элементы которой связаны с элементами матрицы $A = \left\| a_{lc} \right\|$ соотношением $a_l = \sum_{(c)} a_{lc}$, называется μ-свернутой матрицей и обозначается ${}^{\mu}A = \left\| \sum_{(c)} a_{lc} \right\|$. Индексы разбиения *l*=(l_1,…,l_κ) называются свободными индексами, а индексы разбиения *c*=(c_1,…, c_μ)– кэлиевыми индексами.

Пример 2.11. Пусть $A = \left\| a_{i_1 i_2 i_3 i_4} \right\|$ – четырехмерная матрица, все индексы которой принимают значения 1, 2. Если $\kappa = \mu = 2$, $l_1 = i_1$, $l_2 = i_2$ – свободные ин-

дексы и $c_1 = i_3$, $c_2 = i_4$ – кэлиевы индексы, то матрица $^2A = \left\| \sum_{(c)} a_{lc} \right\|$ имеет следующий вид: $^2A = \begin{Vmatrix} \sum_{i_3=1}^{2}\sum_{i_4=1}^{2} a_{11i_3i_4} & \sum_{i_3=1}^{2}\sum_{i_4=1}^{2} a_{12i_3i_4} \\ \sum_{i_3=1}^{2}\sum_{i_4=1}^{2} a_{21i_3i_4} & \sum_{i_3=1}^{2}\sum_{i_4=1}^{2} a_{22i_3i_4} \end{Vmatrix}$.

Сложение. Суммой двух *p*-мерных матриц $A = \left\| a_{i_1 \ldots i_p} \right\|$ и $B = \left\| b_{i_1 \ldots i_p} \right\|$ с одинаковыми наборами индексов $i_1,\ldots,i_p$ называется *p*-мерная матрица $C = \left\| c_{i_1 \ldots i_p} \right\|$ с тем же набором индексов, элементы которой вычисляются по формуле $c_{i_1 \ldots i_p} = a_{i_1 \ldots i_p} + b_{i_1 \ldots i_p}$.

Умножение. Пусть матрицы $A = \left\| a_{i_1 \ldots i_p} \right\|$ и $B = \left\| b_{i_1 \ldots i_q} \right\|$, *p* и *q*-мерные соответственно. Совокупности индексов этих матриц $i_1,\ldots,i_p$ и $i_1,\ldots,i_q$ разбиваются на четыре группы, содержащие соответственно κ, λ, μ и ν индексов (κ, λ, μ, $\nu \geq 0$). Причем $\kappa+\lambda+\mu=p$, а $\lambda+\mu+\nu=q$. Для полученных групп индексов используются обозначения: $l = (l_1,\ldots,l_\kappa)$, $s = (s_1,\ldots,s_\lambda)$, $c = (c_1,\ldots,c_\mu)$ и $m = (m_1,\ldots,m_\nu)$. Тогда матрицы *A* и *B* можно представить в виде в виде $A = \left\| a_{lsc} \right\|$ и $B = \left\| b_{scm} \right\|$. Индексы групп *s* и *c* в матрицах *A* и *B* полностью совпадают. Так же как в операции свертки, индексы разбиения *c* называются кэлиевыми. Индексы разбиения *s* называются скоттовыми, а индексы разбиения *m*, так же как и индексы разбиения *l*, – свободными.

Матрица $C = \left\| c_{lsm} \right\|$, элементы которой вычисляются по формуле $c_{lsm} = \sum_{(c)} a_{lsc} \times b_{scm}$, называется произведением матриц *A* и *B*.

Алгоритм реализации этой операции состоит в следующем:

– перемножаются все пары элементов, у которых полностью совпадают значения индексов групп *s* и *c*,

– суммируются все произведения с одинаковыми значениями индексов группы *c*.

Произведение многомерных матриц называется (λ, μ)-свернутым произведением и обозначается $^{\lambda,\mu}(A\times B)$. Из определения ($\lambda$, μ)-свернутого произведения следует, что для любой пары многомерных матриц можно построить много различных произведений, подбирая различные значения λ и μ.

Число всех возможных (λ, μ)-свернутых произведений p-мерной матрицы A на q-мерную матрицу B вычисляется по формуле $N_{p,q}=\sum_{\lambda+\mu=0}^{\min(p,q)}\frac{p!}{\lambda!\mu!(p-\lambda-\mu)!}\cdot\frac{q!}{\lambda!\mu!(q-\lambda-\mu)!}$.

Пример 2.12. Пример демонстрирует различные варианты (λ, μ)–свернутого произведения двух трехмерных матриц. Пусть дан набор индексов i_1, i_2,i_3, i_4, размерности которых n_1, n_2,n_3, n_4=2. Тогда трехмерные матрицы $\boldsymbol{A}=\left\|\boldsymbol{a}_{i_1i_2i_3}\right\|$ и $\boldsymbol{B}=\left\|\boldsymbol{b}_{i_2i_3i_4}\right\|$ имеют вид:

$$A=\left\|\begin{matrix}a_{111} & a_{112}\\ a_{121} & a_{112}\end{matrix}\right.\left|\begin{matrix}a_{211} & a_{212}\\ a_{221} & a_{222}\end{matrix}\right\| \text{ и } B=\left\|\begin{matrix}b_{111} & b_{112}\\ b_{121} & b_{112}\end{matrix}\right.\left|\begin{matrix}b_{211} & b_{212}\\ b_{221} & b_{222}\end{matrix}\right\|.$$

Среди возможных (λ, μ)–свернутых произведений матриц A и B будут следующие:

– Четырехмерная матрица $C=^{2,0}(A\times B)\Big|^{i_2,i_3}=\left\|c_{i_1i_2i_3i_4}\right\|$, элементы которой вычисляются по формуле $c_{i_1i_2i_3i_4}=a_{i_1i_2i_3}\times b_{i_2i_3i_4}$ для всех совпадающих значений индексов i_2, i_3. Эта матрица имеет вид: $C=\left\|\begin{matrix}a_{111}\times b_{111} & a_{111}\times b_{112}\\ a_{112}\times b_{121} & a_{112}\times b_{122}\end{matrix}\right.\left|\begin{matrix}a_{121}\times b_{211} & a_{121}\times b_{212}\\ a_{122}\times b_{221} & a_{221}\times b_{222}\end{matrix}\right.$
$\left.\begin{matrix}a_{211}\times b_{111} & a_{211}\times b_{112}\\ a_{212}\times b_{121} & a_{212}\times b_{122}\end{matrix}\right|\left.\begin{matrix}a_{221}\times b_{211} & a_{2212}\times b_{2212}\\ a_{222}\times b_{221} & a_{222}\times b_{222}\end{matrix}\right\|$. В этом случае умножение матриц производится без свертки, так как отсутствуют кэлиевы индексы.

– Трехмерная матрица $C=^{1,1}(A\times B)\Big|_{i_2}^{i_3}=\left\|c_{i_1i_3i_4}\right\|$, элементы которой вычисляются по формуле $c_{i_1i_3i_4}=\sum_{i_2=1}^{2}a_{i_1i_2i_3}\times b_{i_2i_3i_4}$. Эта матрица имеет вид:

$$C=\left\|\begin{array}{cc|cc} a_{111}\times b_{111}+a_{121}\times b_{211} & a_{111}\times b_{112}+a_{121}\times b_{212} & a_{211}\times b_{111}+a_{221}\times b_{211} & a_{211}\times b_{112}+a_{221}\times b_{212} \\ a_{112}\times b_{121}+a_{122}\times b_{221} & a_{112}\times b_{122}+a_{122}\times b_{221} & a_{212}\times b_{121}+a_{222}\times b_{221} & a_{212}\times b_{122}+a_{222}\times b_{222} \end{array}\right\|.$$

Трехмерная матрица получается потому, что кэлиев индекс i_2 при свертке удаляется из общей для обеих матриц совокупности индексов.

– Двумерная матрица $C=^{0,2}(A\times B)\Big|_{i_2,\,i_3}=\left\|c_{i_1i_4}\right\|$, элементы которой вычисляются по формуле: $c_{i_1i_4}=\sum_{i_2=1}^{2}\sum_{i_3=1}^{2}a_{i_1i_2i_3}\times b_{i_2i_3i_4}$. Эта матрица имеет вид:

$$C=\left\|\begin{array}{c|c} a_{111}\times b_{111}+a_{111}\times b_{112}+a_{121}\times b_{211}+a_{121}\times b_{212} & a_{211}\times b_{111}+a_{211}\times b_{112}+a_{221}\times b_{211}+a_{221}\times b_{212} \\ a_{112}\times b_{121}+a_{111}\times b_{112}+a_{122}\times b_{221}+a_{122}\times b_{221} & a_{212}\times b_{121}+a_{212}\times b_{122}+a_{222}\times b_{221}+a_{222}\times b_{222} \end{array}\right\|.$$

Определенные таким образом операции над многомерными матрицами позволяют построить, как и в случае файлов, двухосновную алгебраическую систему или АТД, называемый абстрактной алгебраической многомерно-матричной машиной. Особенность построения операций над элементами матриц состоит в том, что в матрицах явно присутствуют нейтральные элементы типа. Поэтому эти операции строятся на основе определенной в 2.1.4 системы функций $f^{j}_{\alpha_1\ldots\alpha_k,\beta_1\ldots\beta_l}(x_{\alpha_1},\ldots,x_{\alpha_k},y_{\beta_1},\ldots,y_{\beta_l})$.

Пусть $\mathscr{M}$ – множество многомерных матриц, а $\mathscr{E}$ – множество их элементов, определяемое для каждой конкретной предметной области. Далее приводится перечень операций, входящих в сигнатуру операций этой алгебраической системы. Унарные операции задают отображение $\mathscr{M}\to\mathscr{M}$, а бинарные – отображение $\mathscr{M}\times\mathscr{M}\to\mathscr{M}$.

Операции структуры	
$\mathcal{T}$	– транспонирование
$\mathcal{S}$	– сечение

C	– свертка
+	– сложение
×	– (λ, μ)-свернутое произведение
Операции типа (над элементами структуры)	
⊕	– аддитивная
⊗	– мультипликативная

В сигнатурах предикатов П включает следующие предикаты:

– z: $\mathscr{M} \to \{0, 1\}$; "матрица X содержит только нейтральные элементы типа;

– ac: $\mathscr{M} \times \mathscr{M} \to \{0, 1\}$; матрицы X и Y совместимы по сложению;

– mc: $\mathscr{M} \times \mathscr{M} \to \{0, 1\}$; матрицы X и Y совместимы по умножению.

Кроме того, в тех случаях, когда элементы матриц – кортежи, на множестве элементов матриц (типе) задаются предикаты *pred*, *eq*, *succ*: $\mathscr{E} \times \mathscr{E} \to \{0, 1\}$, которые задают бинарные отношения *предшествования*, *эквивалентности* и *следования за*.

Тогда абстрактная многомерно-матричная машина задается следующим образом:

$$M_{am} = <\mathscr{M}, \mathscr{E}, \mathscr{T}, S, C, +, \times, \oplus, \otimes; z, ac, mc, pred, eq, succ>.$$

Далее в работе рассмотрены методы построения абстрактной многомерно-матричной машины, основанные на обобщении параллельных алгоритмов, реализующих операции над матрицами на многомерные матрицы.

2.4. Соответствие моделей данных

В этом параграфе устанавливается гомоморфное соответствие между предложенными промежуточными моделями данных и одной из тех моделей данных, которые используются в практике разработки промышленных СУБД и конкретных пользовательских БД.

2.4.1. Соответствие теоретико-множественной и многомерно-матричной моделей

Соответствие операций теоретико-множественной и многомерно-матричной моделей может быть установлено так, как это показано в таблице 2.2.

Таблица 2.2. Соответствие алгебраических операций в теоретико-множественной и многомерно-матричной моделях данных

Теоретико-множественная модель (алгебра файлов)	Многомерно-матричная модель (алгебра многомерных матриц)	Тип операции
сортировка	транспонирование	унарная
выборка	m-кратное сечение	унарная
сжатие	свертка	унарная
слияние строго упорядоченных файлов	сложение	бинарная
слияние нестрого упорядоченных файлов	(λ, μ)-свернутое произведение	бинарная

Таким образом, верно следующее утверждение.

Утверждение 2.2. Сигнатуры операций файловой и многомерно-матричной моделей данных находятся во взаимно однозначном соответствии.

Поскольку рассматривается гомоморфное соответствие алгебр структур данных: таблиц, файлов, многомерных матриц, а не типов их элементов, в дальнейшем для упрощения доказательств, но без ограничения общности, используется алгебра логических многомерных матриц.

Определение 2.2. Многомерная матрица $A = \left\| a_{i_1 \dots i_p} \right\|$ называется логической многомерной матрицей (ЛММ), если ее элементы принадлежат множеству {0, 1} и над ними определены аддитивная операция дизъюнкции и мультипликативная операция конъюнкции.

Операции транспонирования и сечения не зависят от типа элементов матриц, и поэтому их определения остаются без изменений для ЛММ. В определе-

ниях остальных операций изменяются формулы, по которым вычисляются значения элементов ЛММ результата операции.

1. Свертка: если $B=^{\mu}A$, то $b_l = \bigvee\limits_{(c)} a_{lc}$, где l – совокупность свободных, а c – кэлиевых индексов.

2. Сложение: если $C=A+B$, то $c_{i_1..i_p} = a_{i_1..i_p} \circ b_{i_1..i_p}$, где $\circ$ – бинарная логическая операция, которая трактуется как аддитивная операция.

3. (λ, μ)-свернутое произведение: если $C=^{\lambda,\mu}(A\times B)$, то $c_{lsm} = \bigvee\limits_{(c)} a_{lsc} \wedge b_{scm}$.

Утверждение 2.3. Каждому файлу соответствует единственная ЛММ.

Пусть X_K – файл, строго упорядоченный по множеству ключей $K = \{K_1, ..., K_p\}$. Поскольку множества значений ключей конечные, то можно пронумеровать все значения каждого ключа, и те самым поставить в соответствие каждому ключу K_α индекс i_α = (1, …, n_α). Тогда каждому экземпляру множества ключей $K^* = \{K_1^*, ..., K_p^*\}$ соответствует один и только один набор значений индексов $(i_1^0,..., i_p^0)$. Это означает, что между совокупностью всех экземпляров множества ключей K файла X_K и совокупностью всех наборов значений индексов (i_1, …, i_p) установлено взаимно однозначное соответствие. Пусть экземпляру множества ключей $K^* = \{K_1^*, ..., K_p^*\}$ соответствует набор значений индексов $(i_1^0,..., i_p^0)$. Тогда классу эквивалентности X_{K^*} можно поставить в соответствие элемент ЛММ $X = \left\| x_{i_1...i_p} \right\|$, значение которого определяется по формуле:

$$x_{i_1^0..i_p^0} = \begin{cases} 0, \text{ если } X_{K^*} = \Theta, \\ 1, \text{ если } X_{K^*} \neq \Theta. \end{cases}$$

Так как файл X_K строго упорядочен, каждый его класс эквивалентности содержит либо единственную определенную запись, либо универсальную неопределенную запись. Следовательно, при таком методе построения ЛММ каждому строго упорядоченному файлу соответствует единственная ЛММ. Од-

нако, возможна ситуация, при которой нескольким различным файлам, содержащим данные из различных предметных областей, будет соответствовать одна и та же ЛММ. То есть построенное отображение множества строго упорядоченных файлов на множество ЛММ – однозначное.

В следующих трех утверждениях предполагается, что $\boldsymbol{X}$ – множество строго упорядоченных файлов, $\boldsymbol{M}$ – множество ЛММ, $\varphi : \boldsymbol{X} \to \boldsymbol{M}$ – однозначное отображение множества строго упорядоченных файлов на множество ЛММ.

Утверждение 2.4. Если X_K – файл, строго упорядоченный по множеству ключей $K = \{K_1, ..., K_p\}$, а $A = \left\| a_{i_1 \dots i_p} \right\|$ – соответствующая ему ЛММ $(\varphi(X_K) = A)$, и $\begin{pmatrix} K_1, ..., K_p \\ K_{\alpha_1}, ..., K_{\alpha_p} \end{pmatrix}$, $\begin{pmatrix} i_1, ..., i_p \\ i_{\alpha_1}, ..., i_{\alpha_p} \end{pmatrix}$ одинаковые перестановки ключей файла и индексов ЛММ, то $\varphi(sort(X_K)) = A'$, при условии, что сортировка и транспонирование производятся в соответствии с заданными перестановками ключей и индексов.

Это утверждение непосредственно следует из утверждения 2.3 и определений операций сортировки файла и транспонирования многомерной матрицы.

Утверждение 2.5. Пусть X_K – файл, строго упорядоченный по множеству ключей $K = \{K_1, ..., K_p\}$, $A = \left\| a_{i_1 \dots i_p} \right\|$ – соответствующая ему ЛММ $(\varphi(X_K) = A)$, и предикат $\pi(K)$ фиксирует значения ключей K_1, …, $K_m \in K$ ($m<p$), которым соответствуют индексы $(i_1, ..., i_m)$ ЛММ A. Тогда файлу, полученному из файла X_K в результате применения к нему операции выборки в соответствии с предикатом $\pi(K)$, соответствует единственная ЛММ $A_{(i_1, \dots, i_m)}$, полученная из матрицы A m-кратным сечением ориентации $(i_1, ..., i_m)$, то есть $\varphi(sel(X_K, \pi(K)) = A_{(i_1, \dots, i_m)}$.

Достаточно рассмотреть справедливость этого утверждения для случая, когда предикат $\pi(K)$ фиксирует единственный экземпляр каждого из ключей K_1, …, K_m . Тогда каждому фиксированному экземпляру K_j^* ключа K_j соответствует

единственное значение i_j^* индекса i_j (j=1, …, m), и существует простое m-кратное сечение $A_{(i_1,\ldots,i_m)} = \left\| a_{i_1,\ldots,i_p} \right\| \begin{pmatrix} i_1 = i_1^* \\ \cdots \\ i_m = i_m^* \end{pmatrix}$ ЛММ $A = \left\| a_{i_1 \ldots i_p} \right\|$. Если X_{K^*} – класс эквивалентности файла X_K, такой, что в экземпляре множества ключей K^* экземпляры ключей $K_1^*, \ldots, K_m^*$ зафиксированы предикатом $\pi(K)$, а экземпляры остальных ключей – произвольным образом, то этому классу эквивалентности соответствует элемент ЛММ A $a_{i_1^* \ldots i_m^* i_{m+1}^0 \ldots i_p^0} = \begin{cases} 0, \text{ если } X_{K^*} = \Theta_{K^*}, \\ 1, \text{ в противном случае.} \end{cases}$

Но этот элемент будет также принадлежать простому m-кратному сечению $A_{(i_1,\ldots,i_m)}$. Следовательно, всем выбранным классам эквивалентности файла X_K будут соответствовать единственные элементы этого сечения ЛММ A. Вместе с тем, m-кратное сечение можно сконструировать из соответствующих простых m-кратных сечений. Если предикат $\pi(K)$ фиксирует не менее одного экземпляра каждого из ключей K_1, …, K_m , то можно получить все необходимые простые сечения, а затем построить из них необходимую ЛММ, используя операцию сложения ЛММ. Из сказанного следует справедливость утверждения 2.5.

Утверждение 2.6. Пусть X_K – файл, строго упорядоченный по множеству ключей $K = \{K_1, \ldots, K_p\}$, $A = \left\| a_{i_1 \ldots i_p} \right\|$ – соответствующая ему ЛММ ($\varphi(X_K) = A$), $M = \{K_1, \ldots, K_m\}, (m < p)$ – подмножество множества ключей M, по которому файл X_M нестрого упорядочен. Тогда файлу $Y_M = quant(X_M)$ соответствует единственная свертка ЛММ A, то есть $\varphi(quant(X_M)) = {}^{\mu}A$.

В соответствии с определением операции сжатия классы эквивалентности строго упорядоченного файла $quant(X_M)$ получаются по формуле $Y_{M^*} = f(X_{M^*})$, где $f(X_{M^*})$ – функция, реализующая групповую операцию на каждом классе эквивалентности, соответствующем экземпляру множества ключей M^*. То есть,

совокупность записей файла X_K, в которых ключи $K_1, ..., K_m$ имеют одни и те же значения, преобразуется в единственную запись файла Y_M. Каждой записи из этой совокупности соответствует элемент ЛММ A $a_{i_1^* .. i_m^* i_{m+1}^0 .. i_p^0} = 1$. Тогда $a_{i_1^* ... i_m^*} = \bigvee\limits_{(i_{m+1}, ..., i_p)} a_{i_1^* ... i_m^* i_{m+1} ... i_p} = 1$. Следовательно, $\varphi(quant(X_M)) = {}^{\mu}A$.

Утверждение 2.7. Пусть X_K и Y_K – файлы, строго упорядоченные по множеству ключей $K = \{K_1, ..., K_p\}$, $A = \left\| a_{i_1 ... i_p} \right\|$ и $B = \left\| b_{i_1 ... i_p} \right\|$ – соответствующие им ЛММ $(\varphi(X_K) = A$ и $\varphi(Y_K) = B)$. Тогда результату операции слияния строго упорядоченных файлов X_K и Y_K соответствует единственная ЛММ $C = \left\| c_{i_1 ... i_p} \right\|$ такая, что $C=A+B$.

В зависимости от решаемой задачи, для формирования классов эквивалентности (записей) файла – результата слияния строго упорядоченных файлов X_K и Y_K строятся функции, порождающие класс эквивалентности, содержащий либо реальную, либо универсальную неопределенную запись. Аналогично, в качестве аддитивной операции над элементами ЛММ A и B выбирается одна из шестнадцати логических операций, которая формирует элемент ЛММ результата на основе класса эквивалентности файла-результата и по правилам формирования ЛММ – образа файла при отображении φ.

Таким образом, файлу – результату операции слияния строго упорядоченных файлов X_K и Y_K соответствует единственная ЛММ, равная сумме ЛММ A и B с аддитивной операцией $\circ$ над элементами матриц. То есть, $\varphi(ms(X_K, Y_K)) = A +_{(\circ)} B$. Знак операции $+_{(\circ)}$ читается как операция сложения ЛММ с аддитивной операцией $\circ$ над их элементами.

Пример 2.13. Операциям слияния строго упорядоченных файлов, которые реализуют теоретико-множественные операции объединения и пересечения (пример 2.7), соответствуют операции сложения многомерных матриц с аддитивными операциями над элементами: дизъюнкция ($\vee$) и конъюнкция ($\wedge$). Для

операции, реализующей симметрическую разность в качестве аддитивной операции над элементами матриц используется операция "исключающее или" ($\veebar$). Для двух квадратных ЛММ (3×3) это выглядит так:

A	B	$A +_{(\vee)} B$	$A +_{(\wedge)} B$	$A +_{(\veebar)} B$
$\begin{Vmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{Vmatrix}$	$\begin{Vmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{Vmatrix}$	$\begin{Vmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{Vmatrix}$	$\begin{Vmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{Vmatrix}$	$\begin{Vmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{Vmatrix}$

Утверждение 2.8. Пусть X_L и Y_M – файлы, строго упорядоченные по множествам ключей $L=\{L_1, ..., L_p\}$, и $M=\{M_1, ..., M_q\}$ причем выполняется условие $L \cap M \neq \varnothing$, и пусть $K=\{K_1, ..., K_r\}, (r < p+q)$ – множество ключей, связанное с множествами L и M соотношениями: $K \subseteq L \cup M$, $K \cap L \neq \varnothing$, $K \cap M \neq \varnothing$ (файлы $X_{K\cap L}$ и $Y_{K\cap M}$ нестрого упорядочены по своим множествам ключей). $A=\left\|a_{i_1 \dots i_r}\right\|$ и $B=\left\|b_{i_1 \dots i_r}\right\|$ – соответствующие файлам $X_{K\cap L}$ и $Y_{K\cap M}$ ЛММ ($\varphi(X_{K\cap L})=A$ и $\varphi(Y_{K\cap M})=B$). Тогда файлу Z_K – результату операции слияния нестрого упорядоченных по множеству ключей K файлов X_K и Y_K, соответствует единственная многомерная матрица $C=\left\|c_{i_1 \dots i_r}\right\|=^{\lambda,\mu}(A \times B)$.

В соответствии с определением операции слияния нестрого упорядоченных файлов, множество ключей K состоит из трех подмножеств:

1. L' – ключи, принадлежащие только множеству ключей L;
2. M' – ключи, принадлежащие только множеству ключей M;
3. T – ключи, принадлежащие обоим множествам ключей L и M.

При построении операции $^{\lambda,0}(A \times B)$ можно считать, что ключам подмножеств L' и M' соответствуют свободные индексы ЛММ A и B (индексы разбиений l и m), а ключам подмножества T – скоттовы индексы ЛММ A и B (индексы разбиения s). Класс эквивалентности $Z_{K^*} \neq \Theta_{K^*}$ только в том случае, когда соответствующие ему классы эквивалентности $X_{(L'\cup T)^*} \neq \Theta_{(L'\cup T)^*}$ и $Y_{(M'\cup T)^*} \neq \Theta_{(M'\cup T)^*}$. Тогда элементы ЛММ A и B, соответствующие этим классам

эквивалентности, имеют значение 1. А значит, результат конъюнкции этих элементов также будет иметь значение 1. То есть соответствующий этому элементу класс эквивалентности $Z_{K^*} \neq \Theta_{K^*}$. Если функция $f(X_{(K\cap L)^*}, Y_{(K\cap M)^*})$ реализует групповую операцию, то, как показано в параграфе 2.2.3, операция слияния нестрого упорядоченных файлов включает в себя и операцию сжатия файла Z_K, в результате которой получается файл $Z_{K'}$ (множество ключей K' есть подмножество множества ключей K, то есть $K' \subset K$). В этом случае, части ключей подмножества T соответствуют кэлиевы индексы ЛММ A и B (индексы разбиения c), и операции слияния нестрого упорядоченных файлов X_L и Y_M соответствует операция (λ, μ)-свернутого произведения соответствующих им матриц. То есть, $\varphi(mns(X_{K\cap L}, Y_{K\cap M})) = {}^{\lambda,\mu}(A \times B)$.

Утверждения 2.2 – 2.8 позволяют сделать вывод о том, что теоретико-множественная и многомерно-матричная модели данных гомоморфны.

2.4.2. Соответствие промежуточных моделей данных высокоуровневым моделям

Выбор сделан в пользу реляционной модели SQL, поскольку присущий ей язык манипулирования данными – непроцедурный язык SQL ориентирован на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц. То есть предложения этого языка ориентированы в большей степени на конечный результат обработки данных, чем на процедуру этой обработки [70-72]. Таким образом, их суть в наибольшей степени соответствуют сути алгебраических выражений в алгебрах файлов и многомерных матриц. Кроме того, набор операций реляционной модели SQL достаточно полон для решения задач во многих предметных областях. Остальные современные модели данных, либо имеют не такие полные наборы операций, что приводит к необходимости разработки процедур, реализующих недостающие операции, либо, как в объектно-ориентированных моделях [137-138], наборы операций, дополненные средствами разработки пользовательских АТД.

Выбор реляционной модели SQL обусловлен также тем, что промежуточные файловая и многомерно-матричная модели имеют технологическую направленность. Без этой направленности трудно решить проблемы, связанные с распараллеливанием запросов в целом и составляющих эти запросы операций. Одно из требований к данным в этих двух моделях состоит в возможности их упорядочивания по ключам (индексам многомерных матриц). В отличие от классической реляционной модели реляционная модель SQL допускает возможность упорядочивания данных благодаря наличию в операторе SELECT команды ORDER BY. Хотя при записи выражения запроса упорядоченность входных данных не требуется и не учитывается, в силу непроцедурности языка SQL, в технологических целях, можно считать, что операция сортировки в реляционной модели SQL определена. Несмотря на то, что это предположение ослабляет модель, главное ее свойство – ориентированность на конечный результат сохраняется. А в промежуточных моделях, которые являются абстрактными алгебраическими машинами и в которых операции над структурой отделены от операций над ее элементами, это свойство усиливается. Сделанное допущение позволит упростить доказательство соответствия моделей данных.

Как было сказано в параграфе 2.1, для установления соответствия между двумя алгебрами требуется, возможность установления такого взаимнооднозначного соответствия между сигнатурами Ω и Ω', при котором любая операция $F \in \Omega$ и соответствующая ей операция $F' \in \Omega'$ будут n-арными с одним и тем же n.

Таблица 2.3. Соответствие алгебраических операций в реляционной SQL и теоретико-множественной моделях данных

Реляционная модель SQL	Теоретико-множественная модель (алгебра файлов)	Тип операции
SELECT … ORDER BY …	сортировка	унарная
SELECT …WHERE …	выборка	унарная
SELECT … GROUP BY	сжатие	унарная
Теоретико-множественные операции	слияние строго упорядоченных файлов	бинарная

SELECT … FROM JOIN(…)	слияние нестрого упорядоченных файлов	бинарная

Под операциями в реляционной модели SQL будут пониматься запросы, схемы которых представлены в первом столбце таблицы 2.3. Соответствие операций реляционной модели SQL и теоретико-множественной модели может быть установлено так, как это показано в таблице 2.3.

Примеры различных видов операции слияния строго упорядоченных файлов рассмотрены в разделе 2.2.3. Далее приводятся примеры реализации теоретико-множественных операций в реляционной модели SQL.

Объединение:

SELECT $R.A_1$, …, $R.A_n$ FROM R UNION SELECT $S.A_1$, …, $S.A_n$ FROM S;

Пересечение:

SELECT $R.A_1$, …, $R.A_n$ FROM R, S WHERE $R.A_1=S.A_1$ AND … AND $R.A_n=S.A_n$;

Разность:

SELECT $R.A_1$, …, $R.A_n$ FROM R WHERE NOT EXIST

(SELECT $S.A_1$, …, $S.A_n$ FROM S WHERE $R.A_1=S.A_1$ AND … AND $R.A_n=S.A_n$);

Таким образом, верно следующее утверждение.

Утверждение 2.9. Сигнатуры операций реляционной SQL, файловой и многомерно-матричной моделей данных находятся во взаимно-однозначном соответствии.

Следовательно, эти модели данных можно рассматривать как однотипные универсальные алгебраические системы.

Кроме того, в дальнейшем реляционная модель SQL рассматривается, в предположении, что все таблицы (отношения), находятся в третьей нормальной форме. Считается, что третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается [72]. Таблица в третьей нормальной форме, как и строго упорядоченный файл, для каждого определенного значения ключа содержит единственную однозначно определенную строку, неключевые атрибуты которой никаким образом не могут быть

определены посредством других атрибутов этой таблицы. Следовательно, можно утверждать, что каждой таблице, находящейся в третьей нормальной форме, соответствует (с точностью до порядка следования записей) единственный строго упорядоченный файл.

Определения операций над таблицами в реляционной модели SQL [73, 75] аналогичны определениям операций в теоретико-множественной модели. Поэтому можно, не прибегая к сложным формальным доказательствам, считать, что если для соответствующих друг другу реляционной SQL операции и операции алгебры файлов выбраны соответствующие друг другу таблицы и файлы операнды, то и результаты этих операций также будут соответствовать друг другу. При таком построении моделей, с учетом соответствия операций, можно считать, что между реляционной моделью SQL и теоретико-множественной моделью можно установить даже изоморфное соответствие. Из этого следует и гомоморфизм алгебры многомерных матриц и реляционной модели SQL.

2.5. Аксиоматический подход к формализации моделей данных для МОД

В этом разделе рассматривается аксиоматический метод формализации массовой обработки данных (МОД). Аксиоматическому методу для формализации уделено достаточно много внимания исследователей, например, в работах [139-142].

Аксиоматический метод позволяет составлять тексты, формализация которых легко достижима [143, 144]. С инженерной точки зрения, а программирование – это вид инженерной деятельности, использование формализованных текстов открывает большие возможности.

1. Первоначально решение однотипных задач осуществляется в разных организациях с помощью искусственных приемов, которые в процессе обучения передаются от одного программиста другому по принципу «делай как я». Формализация выводит эту деятельность на технический уровень, когда для решения проблемы достаточно определить ее тип, чтобы использовать фор-

мально описанные стандартные алгоритмы (или реализованные в программных библиотеках).

2. Свойства, присущие одной модели общей аксиоматической теории, могут быть использованы для изучения решения аналогичных задач в другой модели. Например, в файлово-ориентированной (теоретико-множественной) модели данных было дано строгое определение файла и строгие определения операций над файлами. Это позволило установить соответствие между этой моделью и многомерной матричной моделью данных, что, в свою очередь, позволило сформулировать принцип симметричного горизонтального распределения для параллельной реализации операции объединения нестрого упорядоченных файлов. Соответствие файловой и реляционной моделей данных обеспечило применение этого принципа для параллельной реализации операции реляционного объединения [145-147]. Это стало возможным благодаря тому, что все три модели данных являются моделями одной аксиоматической теории, которая рассматривается в статье.

2.5.1. Соответствие аксиоматического и алгебраического подходов к формализации МОД

С инженерной точки зрения аксиоматический метод – не что иное, как способ обобщения и формального описания практических (искусственных) методов для решения проблем МОД. Основы для применения этого подхода были заложены в [62, 63]. В этих работах впервые реализована формализация обработки данных на основе алгебраического метода. Как аксиоматический, так и алгебраический методы используются для описания объектов предметной области и операций преобразования одного объекта в другой. В случае МОД объектами являются агрегаты данных: отношения (таблицы), файлы, многомерные матрицы и операции над ними: выбор (раздел), различные операции слияния (сложение и умножение матриц). Основное отличие аксиоматического метода от алгебраического метода заключается в следующем:

Аксиоматический метод используется для формализации операции преобразования объектов путем описания объекта-результата преобразования

исходных объектов. То есть он предоставляет описание объектов и операций их преобразования в соответствии с принципом: "что должно быть результатом операций над объектами-операндами, независимо от того, какова реальность этих объектов и как эта операция выполняется".

Алгебраический метод позволяет конструктивно описывать структуры объектов и алгоритмы операций. Структура объектов определяется либо строго, с использованием математической терминологии, либо по аналогии с известными структурами, например, в файловой модели файл определяется на языке теории множеств, а в реляционном отношении подобный объект имеет вид установить в соответствии с таблицей. Алгоритмы также определяются либо посредством описания на естественном языке, либо с использованием инструментов формализации, присущих теоретическому и практическому программированию. То есть принцип присущ алгебраическому методу: "после применения операции в соответствии с известным алгоритмом к объекту (объектам) с известной структурой получается объект-результат с заданной структурой".

Например, реляционное исчисление есть ни что иное, как аксиоматическое описание объектов (отношений) и операций преобразования одного отношения в другое. Это позволяет определить какое отношение должно быть получено в результате применения заданной операции над отношениями- операндами. В реляционной алгебре объект структурируется как отношение в виде таблиц, а операции над отношениями, как правило, даются в виде словесных (интуитивных) описаний [70-72, 133]. В многомерной-матричной алгебре, которая гомоморфна реляционной алгебре [128, 146-147], каждому отношению соответствует многомерная матрица, которую можно считать высокоструктурированным объектом. Для всех операций приведены строгие формальные алгоритмы. Описания этих операций имеют простые представления в виде: формул, блок-схем, псевдокодов, языков программирования.

2.5.2. Определение аксиоматической теории МОД

Далее рассмотрена формализация МОД для теоретико-множественной (файловой) реляционной, и многомерно-матричной моделей данных. В дальнейшем все они будут считаться единой формальной (аксиоматической) теорией $\mathscr{T}$. Она будет считаться определенной [148, 149], если выполнены следующие условия:

– задан определенный конечный (счетный) набор символов, которые называются символами теории. $\mathscr{T}$;

– определены конечные последовательности символов теории, образующие множество $\mathscr{T}^*$, которые называются выражениями теории $\mathscr{T}$.

Множество символов аксиоматической теории $\mathscr{T}$ для МОД [151, 152] содержит два типа символов:

1. цифры: 0, …, 9;

   заглавные и строчные буквы: a, b, …, A, B, …;

   буквы с индексами: a_1, a_2, …, A_1, A_2, …

2. Специальные символы двух классов:

   a) **термы** (***объекты***) теории $\mathscr{T}$, которые определяются как буквы или последовательности букв;

   b) **формулы** (***соотношения***) теории $\mathscr{T}$.

Интуитивно, термы – это обозначения объектов, а формулы – обозначения утверждений о том, что мы можем сделать с этими объектами. Обычно существует эффективная процедура, которая позволяет для каждого выражения определить, является ли оно формулой. В рассматриваемой теории эта процедура задается рекурсивным соотношением:

1. Любой терм есть формула;

2. A_1, …, A_n), (a_1, …, a_n) – формулы;

3. $B \subset A$, $a \in A$ – формулы;

4. $\forall$ (a_1, …, a_n), $\exists$ (a_1, …, a_n), $f(A_1, \ldots, A_n)$, $f_{a_1,\ldots,a_k}(A_{k+1}, \ldots, A_n)$ – формулы;

5. $R(A_1, \ldots, A_n)$, $\Pi(A_1, \ldots, A_n)$, $\mathscr{K}(A_1, \ldots, A_k)$ – формулы;

6. Если F_1, F_2, F_3 – формулы в смысле п. 5, то ${}^{so}(F_3F_1)$, ${}^{nso}(F_3F_1)$, ${}^{sel}(F_1F_2)$, ${}^{gr}({}^{nso}(F_3F_1))$, ${}^{so}(F_3F_1)\oplus{}^{so}(F_3F_2))$, ${}^{nso}(F_3F_1)\otimes{}^{nso}(F_3F_2))$ – формулы;

7. Если G, G_1, G_2 – формулы в смысле п. 6, and F_1, F_2, F_3 – формулы в смысле п.п. 1-5, то ${}^{so}(F_3G)$, ${}^{nso}(F_3G)$, ${}^{sel}(GF_2)$, ${}^{gr}({}^{nso}(F_3G))$, ${}^{so}(F_3G_1)\oplus{}^{so}(F_3G_2))$, ${}^{nso}(F_3G_1)\otimes{}^{nso}(F_3G_2))$ – формулы;

8. $\{F_1\}$ and $\{F_1|F_2\}$ – формулы.

9. Других формул нет.

2.5.3. Интерпретация формул аксиоматической теории МОД

Определенные таким образом формулы могут интерпретироваться по-разному в разных моделях теории $\mathscr{T}$. Это теоретико-множественная, (файловая), многомерно-матричная и реляционная модели данных.

В этих моделях данных существуют не только терминологические различия, но также существуют различные способы представления объектов и различные алгоритмы операций их преобразования. Список 1 содержит интерпретации в различных моделях данных некоторых формул, которые будут использоваться при построении аксиом.

Таблица 2.4. *Примеры интерпретации формул*

Формулы и их значения в моделях данных		
Теоретико-множественной	Многомерно-матричной	реляционной
A_i, a_i		
поле записи, значение поля	*множество (тип элементов), значение элемента*	*атрибут, значение атрибута*
$(a_1, \ldots, a_n)$		
экземпляр записи файла;	*элемент многомерной матрицы*	*кортеж*
$\mathscr{K}(A_1, \ldots, A_k)R(A_1, \ldots, A_k, A_{k+1}, \ldots, A_n)$		
$X_K(F_1, \ldots, F_n)$ *– файл с записью типа* $(F_1, \ldots, F_n)$ *и множеством ключей* $K=(F_1, \ldots, F_k)$;	*M – многомерная матрица с индексами* $i_1, \ldots, i_k$;	*отношение* $R(A_1, \ldots, A_n)$ *со схемой* $(A_1, \ldots, A_n)$ *и составным ключом* $A_1, \ldots, A_k$.
$f(A_1, \ldots, A_n)$		
n-арная операция над полями записи файла;	*аддитивная операция над элементами многомерной матрицы*;	*n-арная операция, определенная на доменах атрибутов* $A_1, \ldots, A_n$

$f_{a_1,\ldots,a_k}(A_{k+1},\ldots,A_n)$		
n-арная операция над полями группы записей файла с одним и тем же экземпляром множества ключей	*мультипликативная операция над многомерными матрицами с кэлиевыми индексами* $i_1, \ldots, i_k$	*n-арная групповая операция, определенная на доменах атрибутов* $A_{k+1}, \ldots, A_n$.
$^{so}(\mathscr{K}(A_1,\ldots,A_k)R(A_1,\ldots,A_k,A_{k+1},\ldots,A_n))$		
Формулы и их значения в моделях данных		
Многомерно-матричной	Многомерно-матричной	Многомерно-матричной
файл X_K *строго упорядочен по множеству ключей* $K=(F_1, \ldots, F_k)$	*единственный элемент многомерной матрицы соответствующий индексам* $i_1, \ldots, i_k$	*отношение R во второй или третьей нормальной форме с составным ключом* $A_1, \ldots, A_k$
$^{nso}(\mathscr{K}(A_1,\ldots,A_k)R(A_1,\ldots,A_k,A_{k+1},\ldots,A_n))$		
файл X_K *нестрого упорядочен по множеству ключей* $K=(F_1, \ldots, F_k)$	*сечение многомерной матрицы по совокупности индексов* $i_1, \ldots, i_k$	*отношение R в первой нормальной форме с составным ключом* $A_1, \ldots, A_k$

2.5.4. Аксиомы теории массовой обработки данных

Выделен определенный набор формул, называемых аксиомами (или схемами аксиом) теории $\mathscr{T}$. Формулы, которые являются аксиомами для массовой обработки данных, рассматриваются далее.

(A1) Аксиома строгой упорядоченности

$$^{so}(\mathscr{K}(A_1,\ldots,A_k)R(A_1,\ldots,A_k,A_{k+1},\ldots,A_n)) =$$
$$\{(a_1,\ldots,a_k,a_{k+1},\ldots,a_n)\,|\,\forall(a_1,\ldots,a_k,a_{k+1},\ldots,a_n)\neg(\exists(a_1,\ldots,a_k,a'_{k+1},\ldots,a'_n))$$

Эта аксиома означает, что все объекты типа $(a_1,\ldots,a_k,a_{k+1},\ldots,a_n)$ находятся в объекте *R* не более одного раза. Он дает описание агрегата данных как множества.

(A2) Аксиома нестрогой упорядоченности

$$^{nso}(\mathscr{K}(A_1,\ldots,A_k)R(A_1,\ldots,A_k,A_{k+1},\ldots,A_n)) =$$
$$\{(a_1,\ldots,a_k,a_{k+1},\ldots,a_n)\,|\,\forall(a_1,\ldots,a_k,a_{k+1},\ldots,a_n)\exists(a_1,\ldots,a_k,a'_{k+1},\ldots,a'_n)$$

Эта аксиома означает, что объекты типа $(a_1,\ldots,a_k,a_{k+1},\ldots,a_n)$ могут встречаться в объекте *R* более одного раза, то есть совокупность данных определяется как фактормножество (мультимножество). Интерпретации аксиом A1 и A2 в моделях данных приведены в таблице 2.4.

(А3) Аксиома выборки

$$^{sel}R(A_1,...,A_k,A_{k+1},...,A_n)\Pi(a_1,...,a_k) = \{(a_1,...,a_k,a_{k+1},...,a_n) \in (A_1,...,A_k,A_{k+1},...,A_n) \mid \Pi(a_1,...,a_k)\}$$

Эта аксиома дает описание подмножества агрегата данных, элементы которого удовлетворяют условию, определенному предикатом $\Pi(a_1,...,a_k)$. Это подмножество формируется в результате применения операции выборки $^{(sel)}$ к агрегату данных $R(A_1,...,A_k,A_{k+1},...,A_n)$. В файловой модели данных ей соответствует операция выборки, в многомерно-матричной модели данных – операция сечения многомерной матрицы, в реляционной модели данных – оператор SELECT.

(А4) Аксиома сжатия

$$^{gr}(^{nso}(\mathscr{K}(A_1,...,A_k)R(A_1,...,A_k,B_{k+1},...,B_m))) = \{(a_1,...,a_k, f_{a_1,...,a_k}(B'_{k+1},...,B'_m)) \mid B'_{k+1} \subset B_{k+1},...,B'_m \subset B'_m\}$$

Эта аксиома дает описание агрегата данных, который должен быть получен из агрегата данных $R(A_1,...,A_k,B_{k+1},...,B_n)$ после применения операции $f_{a_1,...,a_k}(B'_{k+1},...,B'_m)$. Эта функция применяется ко всем группам его элементов, в которых значения переменных $A_1,...,A_k$ совпадают. Новый агрегат данных формируется в результате применения операции $^{(gr)}$ к агрегату данных $R(A_1,...,A_k,B_{k+1},...,B_n)$. В теоретико-множественной модели данных этой аксиоме соответствует операции сжатия нестрого упорядоченного файла, в многомерно-матричной – свертка, в реляционной – *select ... group by ...* к отношению в первой нормальной форме.

(А5) Аксиома слияния при строгой упорядоченности агрегатов данных

$$^{so}(\mathscr{K}(A_1,...,A_k)R_1(A_1,...,A_k,B_{k+1},...,B_m)) \oplus ^{so}(\mathscr{K}(A_1,...,A_k)R_2(A_1,...,A_k,C_{k+1},...,C_n)) = \{(a_1,...,a_k, f(b_{k+1},...,b_m,c_{k+1},...,c_n))\}$$

Эта аксиома дает описание агрегата данных, который должен быть получен из агрегатов данных $R_1(A_1,...,A_k,B_{k+1},...,B_n)$ и $R_2(A_1,...,A_k,C_{k+1},...,C_n)$ после

применения операции $f(b_{k+1},...,b_m,c_{k+1},...,c_n)$ к их элементам с одинаковыми значениями $a_1,...,a_k$. Новый агрегат данных формируется в результате применения операции ($\oplus$) к этим агрегатам данных. В файловой модели данных она соответствует операции слияния строго упорядоченных файлов, в многомерно-матричной – матричное сложение, в реляционной – любая из теоретико-множественных операций над отношениями, между которыми установлено соответствие один-к-одному.

(А6) Аксиома слияния при нестрогой упорядоченности агрегатов данных

$$^{nso}(\mathcal{K}(A_1,...,A_k)R_1(A_1,...,A_k,B_{k+1},...,B_m)) \otimes {}^{nso}(\mathcal{K}(A_1,...,A_k)R_2(A_1,...,A_k,C_{k+1},...,C_n)) =$$
$$\{(a_1,...,a_k, f_{a_1,...,a_k}(B'_{k+1},...,B'_m,C'_{k+1},...,C'_n))$$
$$\left|B'_{k+1} \subset B_{k+1},...,B'_m \subset B'_m, C'_{k+1} \subset C_{k+1},...,C'_n \subset C_n\right\}$$

Эта аксиома дает описание агрегата данных, который должен быть получен из агрегатов данных $R_1(A_1,...,A_k,B_{k+1},...,B_n)$ и $R_2(A_1,...,A_k,C_{k+1},...,C_n)$ после применения операции $f_{a_1,...a_k}(B'_{k+1},...,B'_m,C'_{k+1},...,C'_n)$ к группам их элементов с одинаковыми значениями $a_1,...,a_k$. Новый агрегат данных формируется в результате применения операции ($\otimes$) к этим агрегатам данных. В файловой модели данных она соответствует операции слияния нестрого упорядоченных файлов, в многомерно-матричной (λ, μ)-свернутому произведению, в реляционной – операции соединения (SELECT… FROM R1 JOIN R2 ON…)) отношений, между которыми установлены отношения один-ко-многим или многие-ко-многим.

Таким образом, используя эти аксиомы, построенные с использованием определений (формулы, приведенные в пунктах 1-5 параграфа 2.5.2 и интерпретированные в таблице 2.4), строятся описания свойств агрегатов данных и операций с ними.

Замечание. Аксиомы А1 и А2 вводят понятия строгой и нестрогой упорядоченности агрегата данных. Они используются далее в аксиомах операций (2-6). В некоторых моделях, например, реляционных, упорядочение агрегатов

данных считается избыточным и не используется. С другой стороны, введение концепции ключа в этой модели требует указания того, будет ли отношение с определенным на нем ключом множеством или мультимножеством. Эта проблема решается с помощью теории нормализации. В файловых и многомерно-матричных моделях, которые используются в качестве промежуточных между моделями проектирования и моделями вычислений, упорядочение агрегатов данных имеет важное значение, поскольку оно определяет природу алгоритмов работы.

2.5.5. Теоремы аксиоматической теории МОД

Пусть существует конечное множество ρ_1, ..., ρ_n отношений между формулами, называемых правилами вывода. Предположим, что для каждого ρ_i существует натуральное число j и множество, состоящее из j формул: $\{ \mathscr{A}_1, \ldots, \mathscr{A}_j \}$. Причем для каждой формулы $\mathscr{A}$ вопрос о том, что все формулы $\mathscr{A}_j$ находятся в отношении ρ_i с формулой $\mathscr{A}$, имеет эффективное решение. Тогда $\mathscr{A}$ называется прямым следствием заданных j формул по правилу вывода ρ_i, то есть имеется вывод формулы $\mathscr{A}$.

В общем случае, формула $\mathscr{A}$ теории $\mathscr{T}$ есть теорема теории $\mathscr{T}$, если существует вывод в $\mathscr{T}$, в котором последняя формула – $\mathscr{A}$.

В случае аксиоматической теории $\mathscr{T}$ МОД теорема определяется следующим образом. Пусть $R(A_1, \ldots, A_n)$ – формула в смысле пункта 5, $\mathscr{F}$– формула в смысле пункта 5 (раздел 2.5.2). Тогда теоремой теории $\mathscr{T}$ называется конструкция вида $R(A_1, \ldots, A_n)= \mathscr{F}$.

Простые теоремы – это подстановки термов непосредственно в аксиомы. Например, подстановка термов реляционной модели: SELECT, FROM … WHERE … в аксиому (A1) приводит к теореме вида SELECT A_1, …, A_n FROM R WHERE $\Pi(A_1,..., A_k)$.

Ниже приведен пример комплексной теоремы, которая получается путем подстановки различных термов в формулы нескольких аксиом.

$$R(A_1,...,A_p,A_{p+1},...,B_q)) = {}^{sel}({}^{nso}(\mathscr{K}(A_1,...,A_k)R_1(A_1,...,A_k,B_{k+1},...,B_r)) \otimes$$
$${}^{nso}(\mathscr{K}(A_1,...,A_k)({}^{so}(\mathscr{K}(A_1,...,A_l)R_2(A_1,...,A_l,C_{l+1},...,C_s)) \oplus$$
$${}^{so}(\mathscr{K}(A_1,...,A_l)R_3(A_1,...,A_l,D_{l+1},...,D_t))))\Pi(A_1,...,A_p).$$

Эта формула имеет следующие интерпретации в рассматриваемых моделях данных.

В файловой модели данных ей соответствует последовательности операций, полученная путем подстановки констант и термов этой модели в формулу. Эта последовательность операций (технологический процесс) состоит из:

1. слияние строго упорядоченных по множеству ключей $\{A_1,...,A_l\}$ файлов R_2 и R_3;

2. слияние нестрого упорядоченных по множеству ключей $\{A_1,...,A_l\}$ файлов R_1 и файла-результата предыдущей операции;

3. выборка записей из файла-результата предыдущей операции, удовлетворяющих условию $\Pi(A_1,...,A_p)$.

Эта последовательность операций есть теорема.

В многомерно-матричной модели данных агрегаты данных – это *p*-, *k*- и *l*-мерные матрицы $R = \left\| R_{i_1...i_p} \right\|, R_1 = \left\| R_{1,i_1...i_k} \right\|, R_2 = \left\| R_{2,i_1...i_l} \right\|, R_3 = \left\| R_{3,i_1...i_l} \right\|$. Тогда формуле соответствует алгебраическое выражение $R = {}^{k,0}(R_1 \times (R_2 + R_3))\begin{vmatrix} \pi(i_1) \\ ... \\ \pi(i_p) \end{vmatrix}$.

Это алгебраическое выражение также есть теорема.

В реляционной модели данных формуле соответствует запрос вида.

SELECT A_1, …, A_q FROM
(R_1 INNER JOIN (SELECT * FROM R_2 UNION ALL R_3) ON $Eq(A_1, …, A_k)$)
WHERE $\Pi(A_1,...,A_p)$.

$Eq(A_1, …, A_k)$ – это условие для сравнения значений атрибутов A_1, ..., A_k отношения R_1 и объединения отношений R_2 и R_3.

Этот запрос также есть теорема.

2.5.6. Эквивалентность моделей МОД

Аксиоматический подход позволяет просто доказать эквивалентность всех трех рассматриваемых моделей МОД. Эта эквивалентность следует из теоремы эквивалентности теорий [148].

Теорема об эквивалентности теорий (Н. Бурбаки, Теория множеств).

Пусть $\mathscr{T}$ – теория, $\mathscr{A}_1$, …, $\mathscr{A}_n$ – ее явные аксиомы, a_1, …, a_h – ее константы, T_1, …, T_h – ее термы. Если $(T_1|a_1)$ ... $(T_h|a_h)$ $\mathscr{A}_i$, $(i = 1, ..., n)$ являются теоремами теории $\mathscr{T}$ ', и знаки теории $\mathscr{T}$ являются знаками теории $\mathscr{T}$ ', и схемы теории $\mathscr{T}$ являются схемами теории $\mathscr{T}$ ', то если $\mathscr{A}$ теорема теории $\mathscr{T}$, то $(T_1|a_1)$ ... $(T_h|a_h)$ $\mathscr{A}$ есть теорема теории $\mathscr{T}$ '.

Здесь $(T_1|a_1)$ ... $(T_h|a_h)$ – подстановки термов T_1, …, T_h вместо констант a_1, …, a_h в формулах $\mathscr{A}_i$, $(i = 1, ..., n)$ и $\mathscr{A}$.

Интуитивно это означает, что если:

– аксиомы теории $\mathscr{T}$ выражают свойства объектов a_1, …, a_h,

– формула $\mathscr{A}$ выражает свойство, вытекающее из этих аксиом,

– объекты T_1, …, T_h, в теории $\mathscr{T}$' обладают свойствами, выраженными аксиомами теории $\mathscr{T}$,

тогда они имеют свойство, выражаемое формулой $\mathscr{A}$. То есть результаты теории $\mathscr{T}$' применяются в теории $\mathscr{T}$'.

Из этой теоремы и сказанного ранее следует, что все три рассматриваемые модели МОД: файловая, многомерная матрица и реляционная – эквивалентны.

Из сказанного можно сделать вывод о том, что аксиоматический метод позволяет решать чисто технические проблемы, возникающие при обработке больших данных в системах МОД.

Как будет показано далее, применение результатов одной теории к другой позволяет использовать методы синтеза и оптимизации запросов, разработанные для многомерных матричных моделей данных [152-154], в других моделях данных: файловых и реляционных. Параллелизм данных, присущий многомер-

ной матричной модели, становится свойством двух других моделей. Более того, методы распараллеливания операций будут практически одинаковыми во всех моделях.

Не менее важна проблема доказательства правильности запросов. Хорошо известны эффективные методы верификации программ, написанных на языках процедурного программирования [125, 155-157]. Отладка сложных запросов к базе данных, которые включают большое количество многократно вложенных подзапросов, несмотря на кажущуюся простоту, на самом деле, является довольно сложным процессом. Это связано с тем, что запрос, отлаженный для небольшого объема данных, на больших объемах данных может быть выполнен с ошибкой или не выполнен вообще. Современные системы управления базами данных используют языки манипуляции данными, такие как PL/SQL и Transact-SQL, которые включают как процедурные инструменты для управления процессом вычислений, так и запросы SQL. Предложенный аксиоматический метод удобно использовать для верификации программ на этих языках.

Таким образом, можно утверждать, что формальный аксиоматический метод может эффективно использоваться для решения инженерных задач при программировании процедур МОД.

2.5.7. Доказательство эквивалентности моделей МОД по операциям слияния

В этом разделе рассматривается применение аксиоматического метода для доказательства соответствия двух моделей МОД по операциям слияния [158]. Аксиомы слияния можно записать с учетом свойств индексов многомерных матриц и ключей отношений (таблиц). Тогда они примут следующий вид.

Аксиома слияния при строгой упорядоченности агрегатов данных (A5)

$$^{so}(\mathcal{K}(A_1,...,A_k)R_1(A_1,...,A_k,B_{k+1},...,B_m)) \oplus {}^{so}(\mathcal{K}(A_1,...,A_k)R_2(A_1,...,A_k,C_{k+1},...,C_n)) = \{(a_1,...,a_k, f(b_{k+1},...,b_m,c_{k+1},...,c_n))\}$$

Эта аксиома дает описание агрегата данных, который должен быть получен из агрегатов данных $R_1(A_1,...,A_k,B_{k+1},...,B_n)$ и $R_2(A_1,...,A_k,C_{k+1},...,C_n)$ после применения

операции $f(b_{k+1},...,b_m,c_{k+1},...,c_n)$ к их элементам с одинаковыми значениями $a_1,...,a_k$. Новый агрегат данных формируется в результате применения операции ($\oplus$) к этим агрегатам данных. В файловой модели данных она соответствует операции слияния строго упорядоченных файлов, в многомерно-матричной – матричное сложение, в реляционной – любая из теоретико-множественных операций над отношениями, между которыми установлено соответствие один-к-одному, при необходимости с дополнительной группировкой.

Аксиома слияния при нестрогой упорядоченности агрегатов данных (А6)

$$^{nso}(\mathscr{K}(A_1,...,A_k)R_1(A_1,...,A_k,B_{k+1},...,B_m,C_{m+1},...,C_p)) \otimes {}^{nso}(\mathscr{K}(A_1,...,A_k)R_2(A_1,...,A_k,D_{k+1},...,D_n,E_{n+1},...,E_q)) =$$
$$\{(a_{k+1},...,a_m,d_{k+1},...,d_n), f_{a_1,...,a_k}(C'_{m+1},...,C'_p,E'_{n+1},...,E'_q)) \,|\, C'_{m+1} \subset C_{m+1},...,C'_p \subset C_p, E'_{n+1} \subset E_{n+1},...,E'_q \subset E_q\}$$

Эта аксиома дает описание агрегата данных, который должен быть получен из агрегатов данных $R_1(A_1,...,A_k,B_{k+1},...,B_m,C_{m+1},...,C_p)$ и $R_2(A_1,...,A_k,D_{k+1},...,D_n,E_{n+1},...,E_q)$ после применения операции $f_{a_1,...,a_k}(C'_{m+1},...,C'_p,E'_{n+1},...,E'_q)$ к группам их элементов с одинаковыми значениями $a_1,...,a_k$. Новый агрегат данных формируется в результате применения операции ($\otimes$) к этим агрегатам данных. В файловой модели данных она соответствует операции слияния нестрого упорядоченных файлов, в многомерно-матричной (λ, μ)-свернутому произведению, в реляционной – операции соединения (SELECT… FROM R1 JOIN R2 ON…)) отношений, между которыми установлены отношения один-ко-многим или многие-ко-многим.

Операции в многомерно-матричной и реляционной моделях

Здесь рассматриваются операции, соответствующие операциям файловой модели: сложение и умножение многомерных матриц, и реляционные операции им соответствующие.

В реляционной алгебре нет операции, которая соответствует операции слияния строго упорядоченных файлов. Операция UNION не может точно реализовать подобную операцию, поскольку если в двух отношениях-операндах встретятся два полностью совпадающих кортежа, то в отношение-результат будет помещен только один из этих кортежей, а не результат их совместной обра-

ботки. Поэтому в большинстве современных систем управления базами данных имеется операция UNION ALL. Эта операция нарушает теоретико-множественный принцип – "отношение есть множество", так как ее результат – "мультимножество", которое может содержать несколько полностью совпадающих элементов. Используя эту операцию можно сконструировать аналог операции слияния строго упорядоченных файлов, которая будет названа *слияние "строго упорядоченных" таблиц*. Термин "строго упорядоченных" взят в кавычки, поскольку в реляционной теории баз данных сортировка не рассматривается как самостоятельная операция. Заметим, что в технологии систем управления базами данных операция сортировки явно используется в алгоритмах, реализующих основные операции.

Слияние "строго упорядоченных" таблиц. Пусть $A(K_1, \ldots, K_p, W)$ и $B(K_1, \ldots, K_p, W)$ – таблицы, схемы которых совпадают. $K_1, \ldots, K_p$ – составной ключ, по которому эти таблицы находятся в третьей нормальной форме. W – неключевые атрибуты, которые могут быть однотипными кортежами, над которыми определена бинарная операция ω. Также определена групповая (агрегатная) операция Ω над группами значений $W_A \omega W_B$ – $\Omega(W_A \omega W_B)$. Тогда запрос

SELECT *UnionAll*.K_1, …, *UnionAll*.K_p, SUM(*UnionAll*.*W*)
FROM (SELECT * FROM *A* UNION ALL SELECT * FROM *B*) AS *UnionAll*
GROUP BY *UnionAll*.K_1, …, *UnionAll*.K_p

реализует операцию слияние "строго упорядоченных" таблиц.

Inner Join. Здесь рассматривается наиболее распространенный и самый сложный вариант этой операции. Пусть даны таблицы $A(K_1, \ldots, K_p, W)$ и $B(K_1, \ldots, K_q, W)$, схемы которых не совпадают. $K_1, \ldots, K_p$ и $K_1, \ldots, K_q$ – составные ключи, по которым эти таблицы находятся в третьей нормальной форме. W – неключевые атрибуты, которые могут быть не обязательно однотипными кортежами, над которыми определена бинарная операция ω. Также определена групповая (агрегатная) операция Ω над группами значений $W_A \omega W_B$ – $\Omega(W_A \omega W_B)$. Кроме того, часть ключей обеих таблиц совпадает. Без ограничения

общности можно считать, что это ключи K_1, …, K_s ($s< p$, $s< q$). Тогда одному и тому же набору значений ключей K_1, …, K_s может соответствовать несколько строк таблицы, то есть они соответствуют нестрого упорядоченным файлам. Не нарушая общности можно предположить, что ключи, используемые в предикате ON, имеют в обеих таблицах индексы 1, …, s. Если часть остальных ключей таблицы A совпадают с ключами таблицы B, то в список полей оператора SELECT они включаются только один раз.

Следующий запрос демонстрирует общий вид этой операции.

SELECT $A.K_{s+1}$, …, $A.K_p$, $B.K_{s+1}$, …, $B.K_q$, $\Omega(A.W \omega B.W)$

FROM A INNER JOIN B ON $A.K_1$, …, $A.K_s = B.K_1$, …, $B.K_s$

GROUP BY $A.K_{s+1}$, …, $A.K_p$, $B.K_{s+1}$, …, $B.K_q$

Для доказательства соответствия операций необходимо представить формализованное в модели алгебраическое выражение в виде, который формализован в виде схемы аксиомы теории массовой обработки данных. Если полученные формулы совпадают с точностью до обозначений термов и констант, то эти операции соответствуют.

Сложение многомерных матриц и слияние "строго упорядоченных" таблиц. Операцию сложения многомерных матриц можно представить в виде формулы

$$\left\| a_{i_1 \dots i_p} \right\| + \left\| b_{i_1 \dots i_p} \right\| = \left\| c_{i_1 \dots i_p} \right\| . \qquad (2.1)$$

Поскольку в каждой из матриц каждому набору значений индексов соответствует единственный элемент, и число элементов во всех матрицах одинаково, то при подстановке знаков формулы (1) в аксиому А5 получается формула

$$^{so}(\mathscr{K}(i_1,...,i_p)A(i_1,...,i_p,a)) \oplus {}^{so}(\mathscr{K}(i_1,...,i_p)B(i_1,...,i_p,b)) = \{(i_1^*,...,i_p^*, a_{i_1^*,...,i_p^*} + b_{i_1^*,...,i_p^*})\} . \quad (2.2)$$

Здесь a и b – элементы матриц A и B, $i_1^*,...,i_p^*$ – фиксированный набор значений индексов.

При слиянии "строго упорядоченных" таблиц считается, что они логически содержат строки со всеми возможными значениями ключей, то есть, часть

строк может быть универсальной неопределенной строкой (записью) Θ. Тогда приведенному SQL-запросу соответствует формула

$$^{so}(\mathscr{K}(K_1,...,K_p)A(K_1,...,K_p,w_a))\oplus{}^{so}(\mathscr{K}(K_1,...,K_p)B(K_1,...,K_p,w_b))=\{(K_1^*,...,K_p^*,w_a+w_b)\}. \quad (2.3)$$

Формулы (2.2) и (2.3) совпадают с точностью до обозначений. Следовательно, операция сложения многомерных матриц и слияния "строго упорядоченных" таблиц соответствуют друг другу.

Умножение многомерных матриц и операция Join. В операции (λ, μ)-свернутого произведения индексы матрицы A представляют собой последовательность вида $l_1,\dots,l_\kappa, s_1,...,s_\lambda, c_1,...,c_\mu$, матрицы B – $s_1,...,s_\lambda, c_1,...,c_\mu, m_1,...,m_\nu$, а матрицы-результата C – $l_1,\dots,l_\kappa, s_1,...,s_\lambda, m_1,...,m_\nu$. Тогда эту операцию можно представить формулой

$$^{\lambda,\mu}\left(\left\|a_{l_1,\dots,l_\kappa s_1,...,s_\lambda c_1,...,c_\mu}\right\|\times\left\|b_{s_1,...,s_\lambda c_1,...,c_\mu m_1,...,m_\nu}\right\|\right)=$$

$$\left\|c_{l_1,\dots,l_\kappa s_1,...,s_\lambda m_1,...,m_\nu}=\sum_{(c_1,...,c_\mu)}a_{l_1,\dots,l_\kappa s_1,...,s_\lambda c_1,...,c_\mu}\times b_{s_1,...,s_\lambda c_1,...,c_\mu m_1,...,m_\nu}\right\| \quad (2.4)$$

После подстановки из формулы 2.4 в аксиому А6, аналогичной той, что была проделана в предыдущем разделе, она примет вид:

$$^{nso}(\mathscr{K}(c_1,...,c_\mu)A(l_1,\dots,l_\kappa s_1,...,s_\lambda c_1,...,c_\mu,a))\otimes{}^{nso}(\mathscr{K}(c_1,...,c_\mu)B(s_1,...,s_\lambda c_1,...,c_\mu m_1,...,m_\nu,b))=$$
$$\{(l_1^*,\dots,l_\kappa^*,s_1^*,...,s_\lambda^*,m_1^*,...,m_\nu^*),\sum_{(c_1,...,c_\mu)}a_{l_1^*,\dots,l_\kappa^*,s_1^*,...,s_\lambda^*c_1,...,c_\mu}\times b_{s_1^*,...,s_\lambda^*c_1,...,c_\mu m_1^*,...,m_\nu^*}\} \quad (2.5)$$

Здесь полиморфные символы операций Σ и $\times$ означают групповую аддитивную и мультипликативную операции.

Пусть схемы таблиц $A(A_1, \dots, A_u)$ и $B(B_1, \dots, B_v)$ могут быть представлены в виде $A(L_1, \dots, L_k, S_1, \dots, S_l, C_1, \dots, C_m, W)$ и $B(S_1, \dots, S_l, C_1, \dots, C_m, M_1, \dots, M_n, W)$. Здесь атрибуты (столбцы таблицы) $S_1, \dots, S_l$ и $C_1, \dots, C_m$ – общие для таблиц A и B, и атрибуты $L_1, \dots, L_k, S_1, \dots, S_l, M_1, \dots, M_n$ – ключи-кандидаты таблицы-результата запроса, представленного формулой

SELECT $A.\ L_1, \dots, A.L_k, A.S_1, \dots, A.S_l, B.M_1, \dots, B.M_n, \Omega(A.W\omega B.W)$
FROM A INNER JOIN B ON $A.C_1, \dots, A.C_m = B.C_1, \dots, B.C_m$ (2.6)
GROUP BY $A.\ L_1, \dots, A.L_k, A.S_1, \dots, A.S_l, B.M_1, \dots, B.M_n$

После подстановки в аксиому А6 знаков запроса из формулы 2.6, получается формула

$$^{nso}(\mathscr{K}(C_1,...,C_m)A(L_1,...,L_k,S_1,...,S_l,C_1,...,C_m,W))\otimes {}^{nso}(\mathscr{K}(C_1,...,C_m)B(S_1,...,S_l,C_1,...,C_m,M_1,...,M_n,W))= \{(L_1^*,...,L_k^*,S_1^*,...,S_l^*,M_1^*,...,M_n^*),\underset{(C_1,...,C_m)}{\Omega}(A.W\,\omega B.W)\} \quad (2.7)$$

Как и в предыдущем случае формулы 2.5 и 2.7 совпадают с точностью до обозначений.

Тогда из теоремы Н. Бурбаки (раздел 2.5.6) об эквивалентности теорий следует, что алгебраическому выражению в многомерно-матричной модели МОД взаимно однозначно соответствует выражение в реляционной модели. Далее приводится пример, иллюстрирующий установление таких соответствий.

Пример 2.14. Пусть дана реляционная база данных, схемы и содержание таблиц которой приведены в таблице 2.1.

Таблица 2.5. База данных примера

A		**B**			**C**			**D**				**E**		**Res**	
K1	**W**	**K1**	**K2**	**W**	**K2**	**K3**	**W**	**K2**	**K3**	**K4**	**W**	**K4**	**W**	**K1**	**S**
p1	100	p1	e1	6	e1	s1	10	e1	s1	m1	10	m1	3	p1	93300
p2	200	p1	e2	3	e1	s1	5	e1	s1	m2	5	m2	5	p2	133000
		p2	e1	2	e1	s2	7	e1	s2	m4	7	m3	2		
		p2	e2	7	e2	s1	5	e2	s1	m2	5	m4	7		
					e2	s2	1	e2	s2	m3	1				
					e2	s3	2	e2	s3	m1	2				

Значение поля *S* для каждого значения поля K1 вычисляется по формуле $S = A.W \times \sum_{(K2)} B.W \times (C.W + \sum_{(K4)} D.W \times E.W)$. В листинге 1 приведен запрос, реализующий вычисление таблицы Res из таблиц *A, B, C D, E*.

Листинг 1. Вычисление таблицы Res

```
1.  SELECT A.K1, SUM(A.W*Q4.W) FROM A Inner Join
2.    (SELECT B.K1, B.K2, SUM(B.W*Q3.W) AS W FROM B Inner Join
3.     (SELECT UA.K2, UA.K3, SUM(UA.W) AS W FROM
4.       (SELECT C.K2, C.K3, C.W FROM C Union All SELECT Q2.* FROM
5.         (SELECT Q1.K2, Q1.K3, Q1.W FROM
6.           (SELECT D.K2 AS K2, D.K3 AS K3, SUM(D.W*E.W) AS W
7.             FROM D Inner Join E ON D.K4=E.K4
8.             GROUP BY D.K2, D.K3)
```

```
9.            AS Q1)
10.         AS Q2)
11.        AS UA
12.       GROUP BY UA.K2, UA.K3)
13.      AS Q3
14.     ON B.K2=Q3.K2
15.     GROUP BY B.K1, B.K2)
16.    AS Q4
17.    ON A.K1=Q4.K1
18.  GROUP BY A.K1
```

Это запрос реализуется следующим образом:

1. Сначала выполняется вложенный запрос Q1, который включается в подзапрос Q2.
2. Подзапрос Q2 – второй операнд в операции Union All, которая выполняется в подзапросе UA.
3. Затем выполняется подзапрос Q3, который включается в подзапрос Q4 в качестве второго операнда операции Inner Join.
4. Последним выполняется основной запрос, в котором подзапрос Q4 используется в качестве второго операнда операции Inner Join.

Подзапросы Q2 и UA реализуют операцию слияния "строго упорядоченных" таблиц.

Как видно из листинга этот запрос, решающий достаточно простую задачу, достаточно сложен как в написании, так и в преобразовании его с целью оптимизации, как в плане распараллеливания на уровне выражения, так и в плане определения наилучшего порядка выполнения операций. Упрощение может быть достигнуто при записи этого запроса в виде выражения в алгебре многомерных матриц.

Домены ключей K1, K2, K3, K4 – конечные множества. Поэтому можно задать нумерацию их элементов. Тогда этим ключам соответствуют индексы i_1, i_2, i_3, i_4. Таблицам соответствуют матрицы

$$A = \|a_{i_1}\| = \|100, 200\|, B = \|b_{i_1 i_2}\| = \begin{Vmatrix} 6 & 3 \\ 2 & 7 \end{Vmatrix}, C = \|b_{i_2 i_3}\| = \begin{Vmatrix} 10 & 5 & 7 \\ 5 & 1 & 2 \end{Vmatrix}, D = \|d_{i_2 i_3 i_4}\| = \left\| \begin{array}{cccc|cccc} 10 & 5 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \end{array} \right\|,$$

$E = \|e_{i_4}\| = \|5, 2, 7\|$.

Таблице Res соответствует матрица *R*, аналогичная матрице *A.* Выражение, вычисляющее значение матрицы *R* вид $R = {}^{0,1}(A \times {}^{0,1}(B \times (C + {}^{0,1}(D \times E))))$. Очевидно, что это выражение гораздо проще выражения запроса, приведенного в листинге 1. Оно может быть легко преобразовано как вручную, так и, в более сложном случае, посредством специализированного программного обеспечения, которое несложно разработать. Кроме того, методы распараллеливания сложения и умножения многомерных матриц [146, 147] позволяют построить аналогичные методы для соответствующих реляционных операций.

2.6. Заключительные замечания к главе 2

В главе рассмотрены методы формализации алгебраических моделей данных и вычислений. Даны основные определения универсальных многоосновных алгебраических систем и приведены примеры таких систем.

Проведен анализ современных интуитивных подходов к объектно-ориентированному моделированию, проектированию и программированию и осуществлен переход к формальному – алгебраическому подходу к объектно-ориентированным технологиям. На основе этого подхода разработана технология построения алгебраических моделей сложных структур данных – кортежей.

Приведено описание объектно-ориентированной технологии программирования для создания программно-аппаратных комплексов на основе универсальной (абстрактной) алгебраической машины – универсальной двухосновной алгебраической системы.

Сделан анализ известных определений типизированного файла и дается формальное алгебраическое определение файла как факторможества множества однотипных записей по отношению эквивалентности, порожденному множеством ключей. Вводится определение универсальной неопределенной записи, назначение которой состоит в том, чтобы в логическом файле были

представлены все возможные значения экземпляров множества ключей. На этой основе строятся формальные описания всех операций обработки файлов.

На основе теории многомерных матриц Н.П. Соколова разработана алгебраическая модель данных (вычислений). Проведено доказательство гомоморфизма теоретико-множественной и многомерно-матричной моделей, но основе которого доказано соответствие многомерно-матричной и реляционной моделей.

Приведено описание аксиоматической теории МОД и на основе теоремы об эквивалентности аксиоматических теорий доказывается соответствие моделей МОД в том числе для конкретного примера. Основные результаты, полученные в данной главе, были опубликованы в работах [117, 128-130, 145, 146, 150-154, 158].

Глава 3. МЕТОДЫ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ МАССОВОЙ ОБРАБОТКИ ДАННЫХ

3.1. Проблемы повышения эффективности обработки данных

Повышение эффективности обработки данных относится к числу наиболее актуальных проблем в теории и практике баз данных. Обычно в декларативных языках запроса, подобных языку SQL, в формулировках запросов указывается, какими свойствами должны обладать данные, которые хочет получить пользователь, но ничего не говорится о том, как система должна реально выполнить запрос. Проблема в том, чтобы по декларативной формулировке запроса найти или построить программу (такую программу принято называть планом выполнения запроса), которая выполнялась бы максимально эффективно и выдавала бы результаты, соответствующие указанным в запросе свойствам. Более точно, основная трудность состоит в том, что нужно уметь:

1. построить все возможные программы, результаты которых соответствуют указанным свойствам;
2. выбрать из множества этих программ (иначе говоря, найти в пространстве планов выполнения запроса) такую программу, выполнение которой было бы наиболее эффективным [106-110].

Далее в главе рассматриваются два способа повышения эффективности процессов МОД, основанные на предложенных в главе 2 моделях. Будет показано, что эти модели позволяют:

1. использовать известные методы оптимизации для синтеза новых и преобразования уже имеющихся процессов;
2. разработать методы организации данных, обеспечивающие эффективное распараллеливание отдельных операций, составляющих эти процессы.

Далее приводятся два примера, иллюстрирующие известные методы повышения эффективности МОД.

Пример 3.1. Оптимизация на уровне процессов обработки данных. Здесь рассматривается подход, основанный на таком хорошо исследованном методе

оптимизации как динамическое программирование. Этот метод применяется, например, при решении задачи выбора оптимальной последовательности операций для вычисления результата алгебраического выражения, состоящего из операций умножения двумерных (плоских) матриц, с помощью алгоритма, приведенного в [85]. Этот алгоритм может быть обобщен на случай многомерных матриц [117].

Пусть $M=M_1\times\ldots\times M_n$. M, M_1, …, M_n – многомерные матрицы. Для любой пары матриц M_i и M_{i+1} сложность вычисления их произведения, независимо от того последовательно или параллельно реализована операция умножения, зависит от величины $p^l\times p^s\times p^c\times p^m$, где:

– $p^l=\prod_{i=1}^{\kappa} d(l_i)$ – произведение размерностей свободных индексов матрицы M_i;

– $p^s=\prod_{i=1}^{\lambda} d(s_i)$ – произведение размерностей скоттовых индексов матриц M_i и M_{i+1};

– $p^c=\prod_{i=1}^{\mu} d(c_i)$ – произведение размерностей кэлиевых индексов матриц M_i и M_{i+1};

– $p^m=\prod_{i=1}^{\nu} d(m_i)$ – произведение размерностей свободных индексов матрицы M_{i+1}.

На каждом шаге основного цикла алгоритма вычисляется величина w_{ij} – минимальная сложность вычисления частичного произведения $M_i\times\ldots\times M_j$ ($1\le i\le j\le n$). При $i<j$ $w_{ij}=\underset{i\le k<j}{\mathrm{MIN}}(w_{ik}+w_{k+1,j}+p^l_{i-1}\times p^s_k\times p^c_k\times p^m_j)$. Здесь w_{ik} – минимальная сложность вычисления матрицы $M'=M_i\times\ldots\times M_k$, $w_{k+1,\,j}$ – минимальная сложность вычисления матрицы $M''=M_{k+1}\times\ldots\times M_j$, а $p^l_{i-1}\times p^s_k\times p^c_k\times p^m_j$ – сложность вычисления произведения многомерных матриц M' и M'', где:

– $p^{l}_{i-1} = \prod_{i=1}^{\kappa} d(l_i)$ – произведение числа значений свободных индексов матрицы M';

– $p^{s}_{k} = \prod_{i=1}^{\lambda} d(s_i)$ – произведение числа значений скоттовых индексов матриц M' и M'';

– $p^{c}_{k} = \prod_{i=1}^{\mu} d(c_i)$ – произведение числа значений кэлиевых индексов матриц M' и M'';

– $p^{m}_{j} = \prod_{i=1}^{\nu} d(m_i)$ – произведение размерностей свободных индексов матрицы M''.

Такой подход возможен для повышения эффективности процессов МОД, если каким-либо образом получено алгебраическое выражение – модель процесса обработки файлов, состоящее или из одних операций умножения многомерных матриц или из суммы нескольких таких выражений. При этом не имеет значения на языке какой из алгебр-моделей это выражение было записано. Естественно, что, если выражение записано на языке алгебры многомерных матриц, размерности индексов фиксированы и в обработку включаются все элементы матриц, в том числе и нейтральные. В файлах и отношениях записи (строки), содержащие универсальные неопределенные значения физически отсутствуют. Поэтому оптимизацию выражения целесообразно производить перед каждым выполнением запроса, заданного этим выражением. Это возможно потому, что рассмотренный алгоритм и ему подобные алгоритмы имеют полиномиальную сложность. Информация о количестве записей (строк), соответствующих экземплярам множеств ключей, которая необходима для реализации таких алгоритмов оптимизации может содержаться в метаданных, определяющих свойства входящих в алгебраическое выражение файлов (отношений).

Пример 3.2. Оптимизация на уровне операций обработки данных. На современной вычислительной технике повышение эффективности операций об-

работки данных осуществляется, как правило, за счет распараллеливания алгоритмов, реализующих эти операции. Типичным примером такого решения проблемы служит алгоритм Кэннона [102, 159], основанный на формуле Фробениуса для умножения матриц [101]. Обобщение этого алгоритма на случай многомерных матриц сделать не так просто, как для предыдущего алгоритма, поэтому здесь он будет рассмотрен для двумерных (плоских) матриц, а в дальнейшем будет построено его обобщение. Этот алгоритм относится к классу блочных алгоритмов, то есть матрицы операнды разбиваются на $q \times q$ блоков. Для простоты и без ограничения общности предполагается, что матрицы квадратные $n \times n$ и n кратно q. Размер каждого блока равен $k \times k$, $(k = \frac{n}{q})$.В этом случае операция матричного умножения матриц A и B в блочном виде может быть представлена так:

$$\begin{Vmatrix} A_{00} & \dots & A_{0\,q-1} \\ & \dots & \\ A_{q-1\,0} & \dots & A_{q\,q} \end{Vmatrix} \times \begin{Vmatrix} B_{00} & \dots & B_{0\,q-1} \\ & \dots & \\ B_{q-1\,0} & \dots & B_{q\,q} \end{Vmatrix} = \begin{Vmatrix} C_{00} & \dots & C_{0\,q-1} \\ & \dots & \\ C_{q-1\,0} & \dots & C_{q\,q} \end{Vmatrix},$$

$$C_{ij} = \sum_{s=0}^{q-1} A_{is} \cdot B_{sj}, \; (i, j = 0, \dots, q-1).$$

В качестве архитектуры вычислительного комплекса используется двумерная процессорная решетка, в которой процессоры связаны так, как это показано на рисунке 3.1.

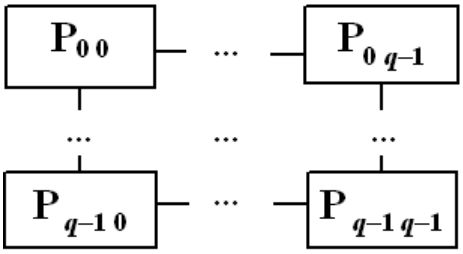


Рис. 3.1. Двумерная решетка $q \times q$ процессоров

На каждом процессоре размещается базовая подзадача, которая выполняет вычисление отдельного блока матрицы C. Алгоритм реализует вычисление произведения матриц в ходе q итераций. На каждой итерации каждая подзадача обрабатывает только по одному блоку исходных матриц A и B. Подзадачи и соответствующие им процессоры нумеруются индексами размещаемых в них

блоков матрицы C, то есть подзадача (i, j) отвечает за вычисление блока C_{ij}. Таким образом, построен программно-аппаратный комплекс, представляющий собой квадратную решетку процессоров, каждый из которых оснащен программным обеспечением, реализующим вычисление соответствующего блока матрицы C. Вся решетка соответствует блочному представлению матрицы C.

Действия по начальной пересылке состоят из следующих шагов:

1. в каждый процесс (i, j) пересылаются блоки A_{ij} и B_{ij}, блок C_{ij} обнуляется;
2. для каждой строки i ($i = 0, 1, ..., q-1$) декартовой решетки процессов выполняется циклический сдвиг блоков матрицы A на i позиций влево (т. е. в направлении убывания номеров столбцов);
3. для каждого столбца j ($j = 0, 1, ..., q-1$) декартовой решетки процессов выполняется циклический сдвиг блоков матрицы B на j позиций вверх (т. е. в направлении убывания номеров строк).

Затем запускается цикл из q итераций, в ходе которого выполняются три действия:

1. содержащиеся в процессе (i, j) блоки матриц A и B перемножаются, и результат прибавляется к блоку матрицы C;
2. для каждой строки i ($i = 0, 1, ..., q-1$) выполняется циклическая пересылка блоков матрицы A, содержащихся в каждом процессе (i, j) этой строки, в направлении убывания номеров столбцов;
3. для каждого столбца j ($j = 0, 1, ..., q-1$) выполняется циклическая пересылка блоков матрицы B, содержащихся в каждом процессе (i, j) этого столбца, в направлении убывания номеров строк.
4. На последней итерации действия (2) и (3) можно не выполнять.

Этот пример показывает, что параллельная реализация операций обработки данных, которая обеспечивает повышение их эффективности за счет уменьшения времени выполнения, тесно связана с архитектурой программно-аппаратного вычислительного комплекса, который реализует эти операции.

Далее будет показано, что теоретико-множественная и многомерно-матричная модели позволяют не только унифицировать набор операций обработки данных, но и применить для решения задачи повышения эффективности процессов их обработки (запросов) различные способы дискретной оптимизации, а для отдельных операций проектировать специализированные программно-аппаратные комплексы.

3.2. Синтез и оптимизация процесса МОД

3.2.1. Модель и метод построения процесса МОД

Модель процесса МОД – это алгебраическое выражение в любой из рассмотренных ранее алгебр: алгебре файлов или алгебре многомерных матриц. В реляционной модели SQL – это выражение запроса, записанное на SQL. Очевидно, что для того, чтобы это выражение имело смысл, необходимо и достаточно, чтобы каждая пара совместно обрабатываемых бинарной операцией файлов (отношений, многомерных матриц) имела общие ключи (для многомерных матриц – это кэлиевы и скоттовы индексы). В дальнейшем будет считаться, что алгебраической моделью процесса служит алгебраическое выражение вида $A_0 = E(A_1, \ldots, A_n)$, правая часть которого состоит из многомерных матриц – моделей исходных файлов, соединенных знаками операций алгебры многомерных матриц, а левая – многомерная матрица – модель выходного файла. При этом предполагается, что существует, по крайней мере, одна перестановка матриц $A_1, \ldots, A_n$, при которой любые две соседние матрицы имеют хотя бы один общий индекс (кэлиев или скоттов), то есть, над ними может быть произведена операция сложения или умножения. Подобная система исходных матриц называется связной. Пример связной системы исходных матриц приведен на рисунке 3.2. В этом графе вершины соответствуют исходным многомерным матрицам, а ребра означают наличие у этих матриц общих кэлиевых и/или скоттовых индексов. При таком подходе умножение многомерных матриц коммутативно с точностью до транспонирования.

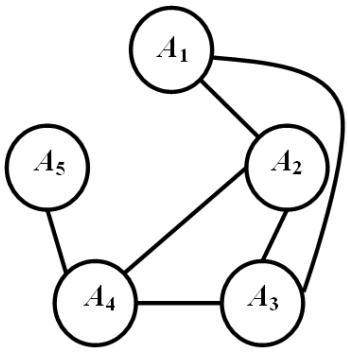


Рис. 3.2. Связная система из пяти многомерных матриц

В общем случае задача заключается в синтезе на основе системы исходных многомерных матриц $A_1, \ldots, A_n$ алгебраического выражения E, в результате вычисления которого получается многомерная матрица A_0. Синтез этого выражения будет возможен только при выполнении двух условий:

1. система исходных многомерных матриц $A_1, \ldots, A_n$ связна;
2. совокупность индексов матрицы A_0 есть подмножество объединения совокупностей индексов матриц $A_1, \ldots, A_n$.

В процессе синтеза алгебраического выражения исходные матрицы выбираются попарно. Чтобы две матрицы могли быть выбраны в качестве операндов одной операции, они должны иметь хотя бы один общий индекс. Если порядок индексов этих матриц не позволяет сразу выполнять предлагаемую в ходе синтеза операцию, то одна или обе матрицы могут быть подвергнуты транспонированию, стоимость которой включается в стоимость синтезированной операции. Выбор операции над матрицами производится по следующим правилам:

1. если все индексы обеих матриц операндов совпадают, выполняется операция сложения матриц;
2. если совпадает только часть индексов, то выполняется операция (λ, μ)-свернутого произведения, те совпадающие индексы, которых нет в матрице-результате и оставшихся матрицах-операндах, включаются в разбиение c (кэлиевы индексы), остальные совпадающие индексы – в разбиение s (скоттовы индексы), а несовпадающие индексы – в разбиения l и m (свободные индексы).

Пример 3.3. Этот пример демонстрирует применение правила 2 для связной системы из пяти исходных матриц, показанной на рисунке 3.2. Пусть:

– A_1 – четырехмерная матрица с набором индексов (i_2, i_3, i_4, i_5);

– A_5 – трехмерная матрица с набором индексов (i_1, i_4, i_5);

– A_{324} – четырехмерная матрица с набором индексов (i_1, i_3, i_4, i_6), полученная в результате применения двух бинарных операций к матрицам A_3, A_2, A_4;

– в результате вычисления выражения $A_0 = E(A_1, A_2, A_3, A_4, A_5)$ должна быть получена трехмерная матрица A с набором индексов (i_1, i_2, i_6).

В этом случае возможно продолжение вычислений тремя различными процессами:

1. $^{0,1}((^{0,1}(A_1 \times A_{324})\big|_{\,i_3}) \times A_5)\big|_{\,i_4}$.
2. $^{0,1}((^{0,1}(A_5 \times A_{324})\big|_{\,i_4}) \times A_1)\big|_{\,i_3}$.
3. $^{0,2}((^{1,1}(A_1 \times A_5)\big|^{\,i_4}_{\,i_5}) \times A_{324})\big|_{\,i_3, i_4}$.

Первые два процесса, практически, одинаковые. В них сначала выполняется операция (0, 1)-свернутого произведения матрицы исходной системы A_1 или A_5 и матрицы A_{324} – результата предыдущих операций. При этом в первом процессе свертка производится по кэлиеву индексу i_3, а во втором – i_4. В результате, в первом процессе получается матрица A_{3241} с набором индексов (i_1, i_2, i_4, i_6), а во втором – матрица A_{3245} с набором индексов (i_1, i_2, i_3, i_6). Следующая операция (0, 1)-свернутого произведения матрицы A_{3241} на матрицу A_5 в первом процессе и матрицы A_{3245} на матрицу A_1 во втором со сверткой по кэлиеву индексу i_4 (i_3) приводит к получению матрицы A – результата вычисления выражения $A = E(A_1, A_2, A_3, A_4, A_5)$. Несмотря на то, что оба процесса очень похожи, время их выполнения может быть различным, так как оно зависит от множеств значений индексов i_3 и i_4.

В третьем процессе сначала выполняется операция (1, 1)-свернутого произведения матриц исходной системы A_1 и A_5. Индекс i_4 в этом произведении используется как скоттов, так как он понадобится при последующем умножении

на матрицу A_{324}, а индекс i_5, который в дальнейшем не понадобится – как кэлиев, и по нему будет выполнена свертка. В результате получается матрица A_{15} с набором индексов (i_1, i_2, i_3, i_4). Операция (0, 2)-свернутого произведения матриц A_{15} и A_{324}, кэлиевы индексы которой i_3 и i_4, приводит к получению матрицы A – результата вычисления выражения $A_0 = E(A_1, A_2, A_3, A_4, A_5)$.

Таким образом, зная наборы индексов всех исходных матриц и набор индексов матрицы-результата можно, применяя предложенный в этом параграфе метод синтезировать алгебраическое выражение – модель процесса МОД.

3.2.2. Формальное описание метода синтеза и оптимизации процесса МОД

Чтобы применить для достижения поставленной в предыдущем разделе цели метод динамического программирования, необходимо доказать, что задача синтеза процесса обработки данных относится к классу динамических оптимизационных задач. Для доказательства используется метод инвариантного погружения [160, 161].

Для решения поставленной задачи в общем виде следует отвлечься от реальных файлов и их моделей, и вместо них рассматривать некоторое множество абстрактных объектов. Предполагается, что эти объекты обладают свойствами, позволяющими собирать их в пары и каждой паре ставить в соответствие некоторое число. Для пояснения дальнейших построений можно считать, что эти объекты – вершины графа, соединенные ребрами, которые означают, что две вершины образуют пару. Для больше ясности все дальнейшие действия будут иллюстрироваться на примере графа, приведенного на рисунке 3.2. Решение обобщенной задачи состоит из следующих действий.

1. Пусть имеется n конечных множеств G. Элементы множества G_s (s=1, … , n) обозначаются a_{i_s} $(i_s = 1, \ldots, n_s)$. Множество G_1 состоит из n элементов, то есть n_1=n. Любым двум элементам $a_{i_1} \in G_1$ и $a_{j_1} \in G_1$ ставится в соответствие вещественное число $c_{i_1 j_1}$. Вводятся следующие обозначения: C_1 – множество чи-

сел $\{c_{i_1 j_1}\}$ $(i_1,\ j_1 = 1, \ldots, n)$ и $S_{i_1 j_1} = c_{i_1 j_1}$. В примере: множество $G_1 = \{a_{1_1}, a_{2_1}, a_{3_1}, a_{4_1}, a_{5_1}\}$; множество $C_1 = \{c_{1_1 2_1}, c_{1_1 3_1}, c_{2_1 3_1}, c_{2_1 4_1}, c_{3_1 4_1}, c_{4_1 5_1}\}$; преобразованный таким образом граф (рисунок 3.2) показан на рисунке 3.4. Из множества C_1 исключены (нейтральные) элементы, которые могли бы соответствовать несуществующим ребрам графа. То есть, элементы множества G_1 соответствуют вершинам графа, а элементы множества C_1 – весам его ребер.

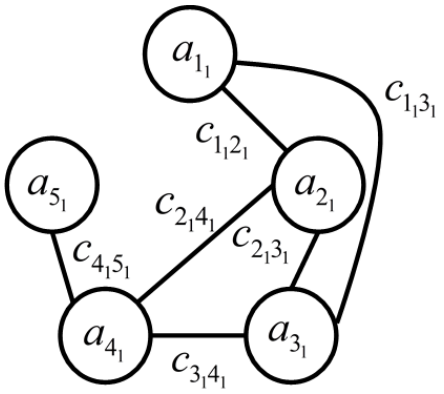


Рис. 3.4. Граф, соответствующий множествам G_1 и C_1

2. Строится множество $G_2[i_1 j_1]$, состоящее из элемента $a_{i_1 j_1}$, который заменяет пару элементов $a_{i_1} \in G_1$ и $a_{j_1} \in G_1$ и всех элементов $a_{k_1} \in G_1$, таких, что $k_1 \neq i_1$ и $k_1 \neq j_1$. Из множества C_1 исключаются все числа $c_{i_1 k_1}$ и $c_{j_1 l_1}$, $(k_1, l_1 = 1, \ldots, n)$ и в него добавляются числа $c_p[i_1 j_1]$ $(1 \le p \le n-2)$, которые ставятся в соответствие элементу $a_{i_1 j_1} \in G_2[i_1 j_1]$ и остальным элементам множества $G_2[i_1 j_1]$. Полученное таким образом множество обозначается $C_2[i_1 j_1]$, а числа из этого множества – $c_{i_2 j_2}[i_1 j_1]$. Элементы множества $G_2[i_1 j_1]$ перенумеровываются и обозначаются $a_{i_2}[i_1 j_1]$. При таком построении множеств $G_2[i_1 j_1]$ и $C_2[i_1 j_1]$ каждой паре элементов $a_{i_2}[i_1 j_1] \in G_2[i_1 j_1]$ и $a_{j_2}[i_1 j_1] \in G_2[i_1 j_1]$ соответствует число $c_{i_2 j_2}[i_1 j_1] \in C_2[i_1 j_1]$. Индексы в квадратных скобках означают, что по некоторому критерию произведен выбор пары объектов a_{i_1} и a_{j_1}. На основе этого выбора произведено преобразование (пересчет по некоторому ал-

горитму) элементов множества C_1 в элементы множества $C_2[i_1 j_1]$. В примере: если из множества G_1 выбраны элементы a_{2_1} и a_{4_1}, и сохранены все связи этих элементов с остальными элементами множества G_1, то будут получены образующие граф (рисунок 3.5) множества $G_2[2_1 4_1] = \{a_{1_2}[2_1 4_1], a_{2_2}[2_1 4_1], a_{3_2}[2_1 4_1], a_{4_2}[2_1 4_1]\}$ и $C_2[2_1 4_1] = \{c_{1_2 2_2}[2_1 4_1], c_{1_2 3_2}[2_1 4_1], c_{2_2 3_2}[2_1 4_1], c_{2_2 4_2}[2_1 4_1], c_{3_2 4_2}[2_1 4_1]\}$. В множестве $G_2[2_1 4_1]$: $a_{1_2}[2_1 4_1] = a_{1_1}$, $a_{3_2}[2_1 4_1] = a_{31}$, $a_{4_2}[2_1 4_1] = a_{5_1}$, $a_{2_2}[2_1 4_1]$ заменяет пару объектов a_{2_1} и a_{4_1}. В множестве $C_2[2_1 4_1]$: $c_{1_2 3_2}[2_1 4_1] = c_{1_1 3_1}$, а остальные элементы вычисляются заново на основе условий взаимодействия нового объекта $a_{2_2}[2_1 4_1]$ с остальными объектами, которые перешли из множества G_1 в множество $G_2[2_1 4_1]$ без изменений. А также $S_{2_1 4_1} = c_{2_1 4_1}$.

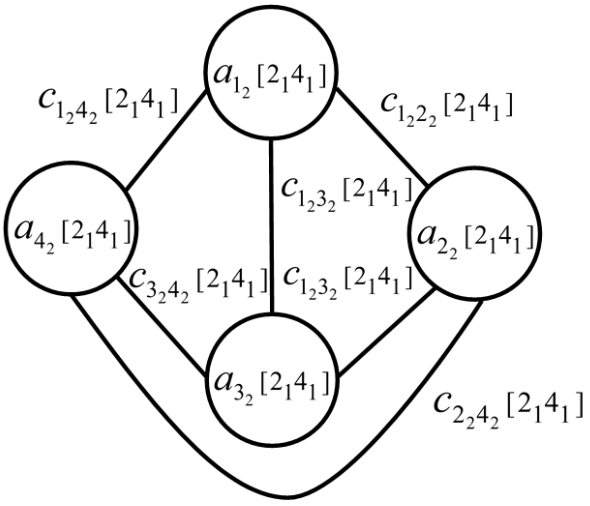


Рис. 3.5. Граф, соответствующий выбору объектов a_{2_1} и a_{4_1}

3. Аналогично строится множество $G_3[i_1 j_1 i_2 j_2]$ элементов $a_{i_3}[i_1 j_1 i_2 j_2]$ и множество $C_3[i_1 j_1 i_2 j_2]$ элементов $c_{i_3 j_3}[i_1 j_1 i_2 j_2], (i_3, j_3 = 1, \ldots, n-2)$ и вводятся обозначения: $S_{i_1 j_1 i_2 j_2} = c_{i_1 j_1} + c_{i_2 j_2}[i_1 j_1]$ и $S_{i_1 j_1 i_2 j_2 i_3 j_3} = c_{i_1 j_1} + c_{i_2 j_2}[i_1 j_1] + c_{i_3 j_3}[i_1 j_1 i_2 j_2]$.

4. Этот процесс продолжается до тех пор, пока не будут получены:

– множество $G_n[i_1 j_1 \ldots i_{n-1} j_{n-1}]$, состоящее из единственного элемента $a_{i_n j_n}[i_1 j_1 \ldots i_{n-1} j_{n-1}], (i_n = j_n = 1)$;

– сумма $S_{i_1 j_1 i_2 j_2 \ldots i_n j_n} = c_{i_1 j_1} + c_{i_2 j_2}[i_1 j_1] + \ldots + c_{i_n j_n}[i_1 j_1 i_2 j_2 \ldots i_{n-1} j_{n-1}]$.

Здесь каждый из индексов i_s и j_s может принимать значения от 1 до n–s+1, s=1, …, n.

Для решения поставленной задачи требуется найти последовательности индексов $\{i_s^0\}$ и $\{j_s^0\}$ такие, что $S_{i_1^0 j_1^0 \ldots i_s^0 j_s^0} \le S_{i_1 j_1 \ldots i_s j_s}$ для всех $\{i_s\}$ и $\{j_s\}$, $(i_s,\ j_s = 1, \ldots, n-s+1)$ отличных от $\{i_s^0\}$ и $\{j_s^0\}$.

Это решение возможно методом динамического программирования. Для чего вводится обозначение $B_s = \min\limits_{\substack{i_s,\ j_s = 1, \ldots, n-s+1 \\ s=1, \ldots, n}} S_{i_1 j_1 \ldots i_s j_s}$ и применяется принцип инвариантного погружения, заключающийся в том, что вместо исходной задачи рассматривается семейство задач $\min \sum\limits_{i=1}^{s} f_i(x_i);\ x_i \in G_i;\ i = 1, \ldots, s;\ s = 1, \ldots, n$. Исходная задача получается при s=n. Применение этого принципа состоит в следующем.

Рассматривается семейство задач минимизации сумм вида $S_{i_k j_k \ldots i_n j_n}[i_1 j_1 \ldots i_{k-1} j_{k-1}] = \sum\limits_{s=k}^{n} c_{i_s j_s}[i_1 j_1 \ldots i_{s-1} j_{s-1}],\ i_s,\ j_s = 1, \ldots, n-s+1,\ s = k, \ldots, n$.

При k=1 получается исходная задача. Далее

$$S_{i_k j_k \ldots i_n j_n}[i_1 j_1 \ldots i_{k-1} j_{k-1}] = c_{i_k j_k}[i_1 j_1 \ldots i_{s-1} j_{s-1}] + \sum_{s=k+1}^{n} c_{i_s j_s}[i_1 j_1 \ldots i_{s-1} j_{s-1}] = c_{i_k j_k}[i_1 j_1 \ldots i_{s-1} j_{s-1}] + S_{i_{k+1} j_{k+1} \ldots i_n j_n}[i_1 j_1 \ldots i_{k-1} j_{k-1}]. \qquad (3.1)$$

Вводится обозначение

$$B_k[i_1 j_1 \ldots i_{k-1} j_{k-1}] = \min_{\substack{i_s,\ j_s = 1, \ldots, n-s+1 \\ s=k, \ldots, n}} S_{i_1 j_1 \ldots i_{s-1} j_{s-1}}. \qquad (3.2)$$

Из равенств (1) и (2) следует, что

$$S_{i_k j_k \ldots i_n j_n}[i_1 j_1 \ldots i_{k-1} j_{k-1}] \geq c_{i_k j_k}[i_1 j_1 \ldots i_{k-1} j_{k-1}] + B_{k+1}[i_1 j_1 \ldots i_{k-1} j_{k-1} i_k j_k] \geq$$
$$\min_{\substack{i_k, j_k = 1, \ldots, n-k+1 \\ k=1, \ldots, n}} (c_{i_k j_k}[i_1 j_1 \ldots i_{s-1} j_{s-1}] + B_{k+1}[i_1 j_1 \ldots i_{k-1} j_{k-1} i_k j_k]).$$

Тогда в силу обозначения (3.2)

$$B_k[i_1 j_1 \ldots i_{k-1} j_{k-1}] = \min_{\substack{i_k, j_k = 1, \ldots, n-k+1 \\ k=1, \ldots, n}} (c_{i_k j_k}[i_1 j_1 \ldots i_{s-1} j_{s-1}] + B_{k+1}[i_1 j_1 \ldots i_{k-1} j_{k-1} i_k j_k]) \quad (3.3)$$

Уравнение (3.3) является уравнением Беллмана для рассматриваемой исходной задачи. Так как $i_n=j_n=1$, граничное условие для уравнения (3) записывается следующим образом

$$B_n[i_1 j_1 \ldots i_{n-1} j_{n-1}] = c_{11}[i_1 j_1 \ldots i_{n-1} j_{n-1}]. \quad (3.4)$$

Пусть $i_k^*(i_1 j_1 \ldots i_{k-1} j_{k-1})$ и $j_k^*(i_1 j_1 \ldots i_{k-1} j_{k-1})$ те значения индексов i_k и j_k при которых достигается минимум в (3.3). При решении уравнения (3.3) с граничными условиями (3.4) справа налево получается выражение

$$B_{n-1}[i_1 j_1 \ldots i_{n-2} j_{n-2}] = \min_{i_{n-1}, j_{n-1} = 1, 2} (c_{i_{n-1} j_{n-1}}[i_1 j_1 \ldots i_{n-2} j_{n-2}] + c_{11}[i_1 j_1 \ldots i_{n-1} j_{n-1}]) =$$
$$= c_{i_{n-1}^0 j_{n-1}^0}[i_1 j_1 \ldots i_{n-2} j_{n-2}] + c_{11}[i_1 j_1 \ldots i_{n-2} j_{n-2} i_{n-1}^0 j_{n-1}^0], \quad (3.5)$$

где $i_{n-1}^0 = i_{n-1}^*(i_1 j_1 \ldots i_{n-2} j_{n-2})$ и $j_{n-1}^0 = j_{n-1}^*(i_1 j_1 \ldots i_{n-2} j_{n-2})$. При подстановке равенства (3.5) в уравнение (3.3) при $k=n-2$, получается выражение

$$B_{n-2}[i_1 j_1 \ldots i_{n-3} j_{n-3}] = \min_{i_{n-2}, j_{n-2} = 1, 2, 3} (c_{i_{n-2} j_{n-2}}[i_1 j_1 \ldots i_{n-3} j_{n-3}] + c_{i_{n-1}^0 j_{n-1}^0}[i_1 j_1 \ldots i_{n-2} j_{n-2}] +$$
$$+ c_{11}[i_1 j_1 \ldots i_{n-2} j_{n-2} i_{n-1}^0 j_{n-1}^0]) = c_{i_{n-2}^0 j_{n-2}^0}[i_1 j_1 \ldots i_{n-3} j_{n-3}] + c_{i_{n-1}^0 j_{n-1}^0}[i_1 j_1 \ldots i_{n-3} j_{n-3} i_{n-2}^0 j_{n-2}^0] +$$
$$+ c_{11}[i_1 j_1 \ldots i_{n-3} j_{n-3} i_{n-2}^0 j_{n-2}^0 i_{n-1}^0 j_{n-1}^0],$$

где $i_{n-2}^0 = i_{n-2}^*(i_1 j_1 \ldots i_{n-3} j_{n-3})$ и $j_{n-2}^0 = j_{n-2}^*(i_1 j_1 \ldots i_{n-3} j_{n-3})$.

При продолжении процесса подстановок, через конечное число шагов будет получено выражение

$$B_1 = \min_{i_1, j_1 = 1, \ldots, n} (c_{i_1 j_1} + B_2[i_1 j_1]) = c_{i_1^0 j_1^0} + B_2[i_1^0 j_1^0] =$$
$$= c_{i_1^0 j_1^0} + \ldots + c_{i_{n-1}^0 j_{n-1}^0}[i_1^0 j_1^0 \ldots i_{n-2}^0 j_{n-2}^0] + + c_{11}[i_1^0 j_1^0 \ldots i_{n-1}^0 j_{n-1}^0],$$

где

$$i_1^0 = i_1^* \text{ и } j_1^0 = j_1^* \quad (3.6)$$

и

$$i_s^0 = i_s^*(i_1 j_1 \ldots i_{s-1} j_{s-1}) \text{ и } j_s^0 = j_s^*(i_1 j_1 \ldots i_{s-1} j_{s-1}), s = 1, \ldots, n. \quad (3.7)$$

Последовательности $\{i_s^0\}$ и $\{j_s^0\}$, где $s = 1, \ldots, n$, определяемые из выражений (3.6) и (3.7) представляют собой искомое решение задачи в общем случае.

Таким образом, доказано, что решение задачи синтеза и оптимизации процесса МОД может быть реализовано с применением метода динамического программирования [74, 117, 162-164].

3.2.3. Реализация метода синтеза и оптимизации процесса МОД

Решение задачи формализации в предыдущем разделе было проведено вне зависимости от конкретного смысла элементов исходного множества и характера взаимоотношений между ними. Теперь необходимо определить условия, соответствующие поставленной в начале параграфа задаче синтеза и оптимизации процесса МОД. Для этого требуется придать конкретный смысл элементам множеств G_s (s=1, … , n). Очевидно, в роли этих элементов должны выступать агрегаты данных любой из рассмотренных ранее моделей данных: файлы, многомерные матрицы, отношения (таблицы). Поскольку эти модели данных есть, по крайней мере, гомоморфные универсальные алгебраические системы, в качестве элементов множеств G_s можно выбрать любой из перечисленных. Разница заключается только в способе оценки стоимости операций. Для многомерных матриц, как было показано в примере 3.1, оценка стоимости операций производится на основе сведений о числе значений индексов многомерных матриц – операндов рассматриваемой операции. В случае вычисления сложности операций над файлами или отношениями, стоимость операции вычисляется на основе метаданных, содержащих информацию о количестве записей (кортежей) в классах эквивалентности для всех экземпляров множества ключей. Эта информация может быть легко получена при индексно-последовательной организации данных.

Пусть элементам множества G_1 соответствуют файлы системы исходных файлов $\{A_1, \ldots, A_n\}$. Тогда числа $c_{i_1 j_1}$, из множества C_1 соответствуют стоимостям операций над файлами A_{i_1} и A_{j_1}. Очевидно, что множества $G_s[i_1 j_1 \ldots i_{s-1} j_{s-1}]$ соответствуют промежуточным системам исходных файлов, а множества $C_s[i_1 j_1 \ldots i_{s-1} j_{s-1}]$ содержат стоимости операций обработки пар файлов из этих промежуточных систем. Тогда первоначальной задаче синтеза алгебраического выражения – модели процесса последовательной обработкой файлов соответствует частный случай решенной выше общей задачи, получаемый при следующих ограничениях:

1. $c_{i_s i_s}[i_1 j_1 \ldots i_{s-1} j_{s-1}] = 0$, то есть любой из исходных файлов может быть только одним операндом в любой из операций (не обрабатывается совместно сам с собой);

2. Выбор пары элементов $a_{i_s}[i_1 j_1 \ldots i_{s-1} j_{s-1}]$ и $a_{j_s}[i_1 j_1 \ldots i_{s-1} j_{s-1}]$ принадлежащих множеству $G_s[i_1 j_1 \ldots i_{s-1} j_{s-1}]$ и замена ее на элемент множества $G_{s+1}[i_1 j_1 \ldots i_s j_s]$ производится только в том случае, когда $c_{i_s j_s}[i_1 j_1 \ldots i_{s-1} j_{s-1}] \neq 0$. Выбор такой пары означает, что над файлами, которые соответствуют элементам пары, выполняется одна из операций совместной обработки (сложение или умножение).

3. $c_{i_s j_s}[i_1 j_1 \ldots i_{s-1} j_{s-1}] = c_{j_s i_s}[i_1 j_1 \ldots i_{s-1} j_{s-1}]$, если над файлами, соответствующими элементам пары, выбранной из множества $G_s[i_1 j_1 \ldots i_{s-1} j_{s-1}]$, производится операция сложения.

Синтез алгебраического выражения – модели оптимального процесса обработки файлов методом динамического программирования реализуется в два этапа.

На первом этапе строится граф-сеть, отражающий все пошаговые переходы от начальной системы исходных файлов к системе, содер-

жащей только файл-результат. В этом графе вершины соответствуют системам исходных файлов, ребра – операциям, переводящим систему из q исходных файлов в систему из q – 1 исходного файла.

На рисунке 3.3 приведен пример подобного графа сети для неполносвязной системы исходных файлов, показанной на рисунке 3.2. Исходные файлы обозначены $A_1, \ldots, A_5$; промежуточные файлы, полученные в результате обработки s исходных файлов – $G_{i_1 \ldots i_s}$ $(s \le 5, i_s \le 5)$.

На первом шаге в ходе построения графа-сети из системы исходных файлов $S = \{A_1, \ldots, A_n\}$ выбираются все допустимые пары (A_i, A_j), $(i, j = 1, \ldots, n)$ и строятся промежуточные системы вида $S_{1k} = \{A_{11}, \ldots, A_{1n-1}\}$, где один из файлов $A_{11}, \ldots, A_{1n-1}$ есть результат операции над файлами $A_i, A_j \in S$.

На этом же шаге строятся и оцениваются связи, соответствующие операциям, переводящим исходную систему в одну из построенных.

Следующий шаг заключается в построении всех возможных систем S_{2k}, состоящих из n – 2 файлов, и установлении связей между системами, построенными на предыдущем шаге, и этими системами. Теперь в каждую систему S_{2k} возможен переход из нескольких систем S_{1k}. Следовательно, в вершину, соответствующую произвольной системе S_{2k}, ведут несколько ребер, исходящих из вершин, соответствующих системам S_{1k}. Оцениваются связи, соответствующие операциям, переводящим системы S_{1k} в системы S_{2k}. Все последующие шаги представляют собой аналогичные построения и продолжаются до тех пор, пока не будут построены все системы, состоящие из двух матриц. На последнем шаге, при переходе из построенных вершин в конечную, строятся все ребра, соответствующие операциям, переводящим системы из двух исходных файлов в файл-результат. Оцениваются соответствующие этим ребрам операции и первый этап завершается.

Второй этап и есть, собственно, динамическое программирование. Здесь на каждом шаге, в каждой вершине выбирается условно-оптимальный процесс,

соответствующий последовательной операции с минимальной суммарной стоимостью. На последнем шаге, который соответствует переходу от системы исходных файлов к системам из $n - 1$ файла, среди условно-оптимальных процессов выбирается оптимальный. На рисунке 3.3 выделены промежуточные системы исходных файлов, которые выбираются на каждом шаге, а оптимальная траектория обозначена пунктирной линией.

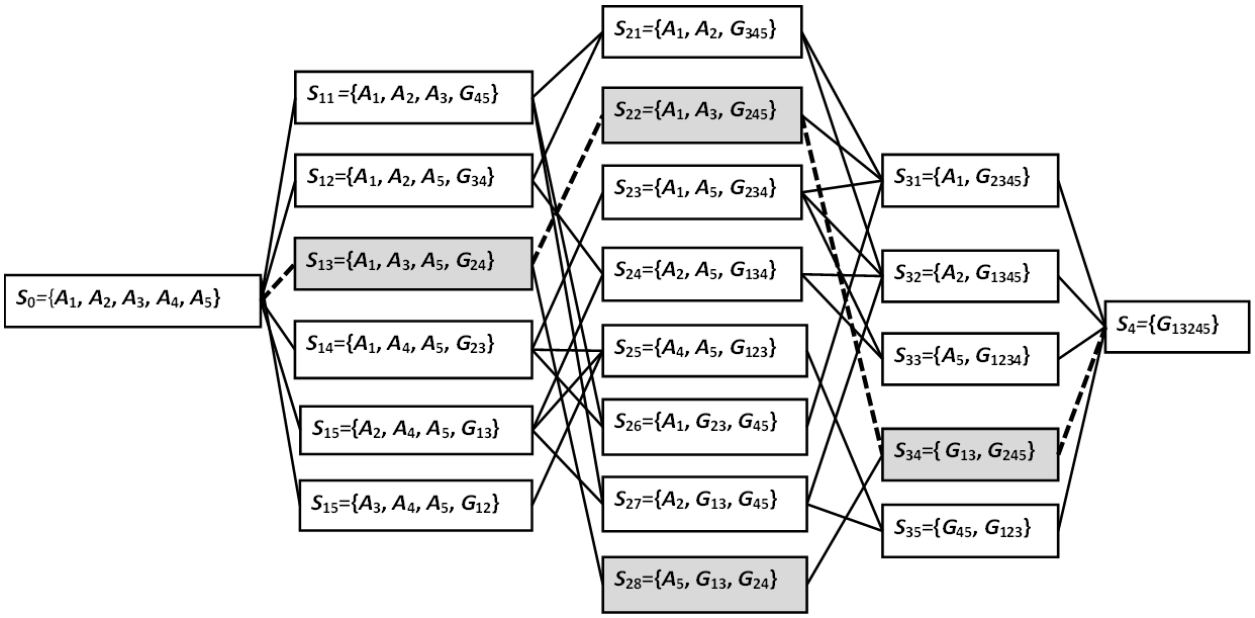


Рис. 3.3. Синтез оптимального процесса МОД методом динамического программирования

Так реализуется алгоритм синтеза процесса обработки файлов методом динамического программирования. Этот метод хорош тем, что его результатом является оптимальный процесс.

Кроме того, полученный процесс может быть улучшен использованием алгоритма ветвей-границ. Считая полученное значение сложности процесса "рекордом" можно привести перебор возможных процессов с целью поиска процесса меньшей сложности. Для доказательства этой возможности был проведен вычислительный эксперимент, в ходе которого строились процессы обработки системы из десяти исходных файлов. Случайным образом генерировались группы по 10000 процессов, и вычислялась стоимость каждого процесса. Были сгенерированы 1000 групп, в которых распределения процессов по слож-

ности отличались незначительно. Результат, полученный для одной из таких групп, приведен на рисунке 3.4.

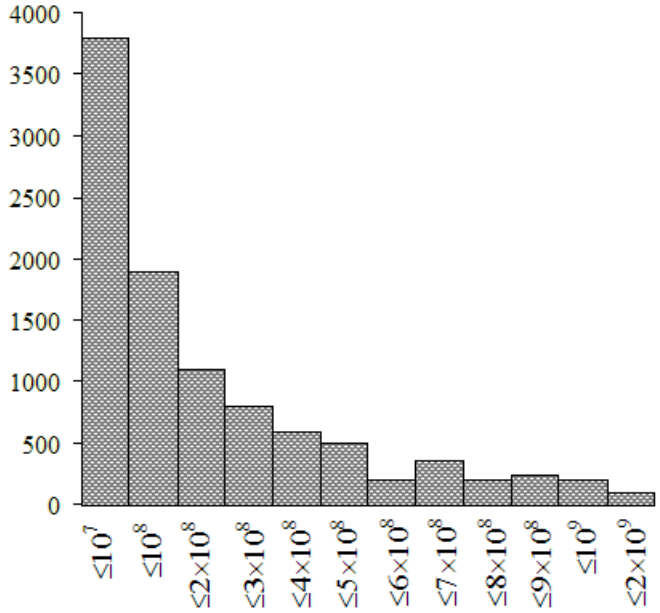


Рис. 3.4. Гистограмма распределения процессов МОД по сложности (суммарные длины просмотров)

3.3. Выбор параллельных алгоритмов для реализации операций МОД

Как было сказано в главе 1, реализация массовой обработки данных должна удовлетворять требованиям максимального параллелизма. Это означает, что программно-аппаратные комплексы, ориентированные на решение задач МОД должны конструироваться таким образом, чтобы можно было реализовать параллелизмы, как на уровне процессов, так и на уровне операций. Как было показано в примерах 1.2 и 1.3, для параллельного решения задачи МОД могут быть построены вычислительные комплексы с различными архитектурами. Причем цель, которую необходимо достигнуть, состоит в том, чтобы разработчик имел возможность построения индивидуального программно-аппаратного комплекса для каждой задачи. В 3.2 было показано, что если задана система исходных агрегатов данных (файлов, отношений, многомерных матриц) A_1, …, A_n, то алгебраическое выражение – модель процесса обработки данных (запроса), приводящее к получению результирующего агрегата A, можно представить в

виде $\bigoplus_{i=1}^{n} A_1 \otimes ... \otimes A_{n_i} (\sum_{i=1}^{n} n_i = n)$, то есть в виде "суммы произведений". $\oplus$ – адди-

тивная (слияние строго упорядоченных файлов, сложение многомерных матриц, любая из теоретико-множественных) операция, $\otimes$ – мультипликативная (слияние нестрого упорядоченных файлов, умножение многомерных матриц, Join) операция. В этом случае возникает возможность комбинировать различные способы распараллеливания вычисления этого выражения, используя распараллеливание, как операций, так и процесса вычислений. При этом возможны различные сочетания методов и алгоритмов последовательной и параллельной обработки данных в ходе вычисления алгебраического выражения. Подобный подход применялся при решении задач линейной алгебры [165-167] как принцип попеременно последовательно-параллельных вычислений, а также при решении проблемы оптимизации запросов к базам данных [168] как метод последовательно-параллельного программирования.

Из сказанного следует, что вычислительный комплекс, ориентированный на решение задач МОД, должен состоять из совокупности соединенных между собой подсистем, каждая из которых параллельно реализует одну из операций, а вместе они реализуют весь процесс вычислений. Причем, в зависимости от характера алгебраического выражения, реализация процесса вычислений может быть как последовательной, так и параллельной. При построении гибких вычислительных комплексов для реализации МОД следует учесть тот факт, что в каждом конкретном случае, архитектура комплекса зависит от выбранного алгоритма, который реализует операцию или процесс обработки данных. Далее будут рассмотрены конкретные алгоритмы для реализации операции слияния нестрого упорядоченных файлов, умножения многомерных матриц или естественного соединения отношений (JOIN), так как этот класс операций имеет наивысшую вычислительную сложность.

Выбор одной из рассмотренных моделей данных для параллельной реализации операций и процессов МОД определяется характером данных в конкретной решаемой задаче. Важную роль в этом выборе играют ключи. Пусть в ис-

ходных файлах каждый ключ принимает все, или почти все, значения из возможных. В этом случае матрицы – модели исходных файлов будут плотными, то есть не будут содержать (или будут содержать незначительное количество) нейтральных элементов, которые соответствуют универсальным неопределенным записям. Естественно, что в таком случае наилучшей будет многомерно-матричная модель. В противном случае, если многомерная матрица получается разреженной [169, 170], использование файловой модели или реляционной модели SQL будет более эффективным.

3.4. Алгоритм параллельного умножения многомерных матриц

3.4.1. Выбор алгоритма умножения многомерных матриц

Для параллельного умножения обычных матриц, как правило, используются два вида алгоритмов [159]: ленточные, реализующие поэлементное умножение матриц и блочные. Многие авторы, проводившие анализ эффективности параллельного умножения матриц [170-173], отдают предпочтение блочным алгоритмам, утверждая, что последние обладают высокой степенью масштабируемости. Учитывая тот факт, что масштабируемость – это наиболее важное свойство параллельных алгоритмов и вычислительных комплексов их реализующих, в дальнейшем будет рассматриваться параллельный алгоритм блочного умножения многомерных матриц. Метод Фробениуса естественно обобщается на многомерные матрицы. Известны различные реализации параллельного умножения матриц: алгоритмы Фокса и Кэннона [171], универсальный алгоритм для параллельных вычислительных комплексов с распределенной памятью PUMMA [171, 172], масштабируемый универсальный алгоритм SUMMA [173]. Для обобщения на многомерные матрицы далее рассматривается алгоритм Кэннона, однако, предложенный метод обобщения может быть распространен и на другие алгоритмы блочного умножения матриц.

Особенность умножения многомерных матриц, отличающая его от умножения обычных (плоских) матриц и тензоров, заключается в наличии скоттовых индексов, которые присутствуют в обеих матрицах-операндах и в матрице-результате [174-177]. Эта особенность существенно сказывается на алгоритме умножения многомерных матриц. При отсутствии скоттовых индексов этот алгоритм мало отличается от алгоритма умножения плоских матриц. Далее рассматривается алгоритм параллельного умножения многомерных матриц при наличии скоттовых и кэлиевых индексов ($\lambda>0$, $\mu>0$) . Для простоты, но без ограничения общности, удобно считать, что количества значений всех индексов во всех разбиениях l, s, c, m матриц-сомножителей A и B одинаковы, то есть равны числу n, которое делится на целое число E.

Пусть даны: p-мерная матрица A (A_{lsc}), q-мерная матрица B (B_{scm}) и r-мерная матрица C (C_{lsm}) ($r=p+q-\lambda-2\mu$) такая, что $C=^{\lambda,\mu}(A\times B)$. Если зафиксировать по одному значению каждого из скоттовых индексов, одинаковому во всех трех матрицах ($s_1^0, ..., s_\lambda^0$), то будут получены: ($p-\lambda$)-кратное сечение матрицы A, ($q-\lambda$)-кратное сечение матрицы B и ($r-\lambda$)-кратное сечение матрицы C. Все эти сечения будут иметь ориентацию $(s_1, ..., s_\lambda)$. Пусть $s_1^1, ..., s_\lambda^1$ и $s_1^2, ..., s_\lambda^2$ фиксированные наборы значений скоттовых индексов многомерных матриц A, B и C, отличающиеся друг от друга, по крайней мере, одним значением. Этим наборам значений скоттовых индексов соответствуют два различных сечения каждой из матриц. Элементы соответствующих сечений матрицы C вычисляются по следующим формулам:

$$c_{l_1...l_\kappa s_1^1...s_\lambda^1 m_1...m_\nu} = \sum_{c_1,...,c_\mu=0}^{n} a_{l_1...l_\kappa s_1^1...s_\lambda^1 c_1...c_\mu} \cdot b_{s_1^1...s_\lambda^1 c_1...c_\mu m_1...m_\nu},$$

$$c_{l_1...l_\kappa s_1^2...s_\lambda^2 m_1...m_\nu} = \sum_{c_1,...,c_\mu=0}^{n} a_{l_1...l_\kappa s_1^2...s_\lambda^2 c_1...c_\mu} \cdot b_{s_1^2...s_\lambda^2 c_1...c_\mu m_1...m_\nu}.$$

Из приведенных формул вытекает следующее утверждение.

Утверждение 3.1. Результат (λ, μ)-свернутого произведения двух многомерных матриц A и B размерностей p и q, при наличии скоттовых индексов (λ>0), есть r-мерная матрица C ($r=p+q-\lambda-2\mu$), составленная из ($r-\lambda$)-кратных сечений, каждое из которых есть произведение ($p-\lambda$)-кратного сечения матрицы A и ($q-\lambda$)-кратного сечения матрицы B. При этом наборы значений индексов $s_1, ..., s_\lambda$ сечений-сомножителей и сечения-результата совпадают.

Следующий пример иллюстрирует приведенное в утверждении 3.1. свойство умножения многомерных матриц.

Пример 3.4. Пусть $A=\|a_{lsc}\|, B=\|b_{scm}\|$ и $C=\|c_{lsm}\|$ трехмерные матрицы, индексы которых принимают значения от 0 до 2. Однократные сечения ориентации s этих матриц представляют собой двумерные матрицы:

$$A_{lsc}=\left\|\begin{array}{ccc|ccc|ccc} a_{000} & a_{001} & a_{002} & a_{010} & a_{011} & a_{012} & a_{020} & a_{021} & a_{022} \\ a_{100} & a_{101} & a_{102} & a_{110} & a_{111} & a_{112} & a_{120} & a_{121} & a_{122} \\ a_{200} & a_{201} & a_{202} & a_{210} & a_{211} & a_{212} & a_{220} & a_{221} & a_{222} \end{array}\right\|,$$

$$B_{scm}=\left\|\begin{array}{ccc|ccc|ccc} b_{000} & b_{001} & b_{002} & b_{100} & b_{101} & b_{002} & b_{200} & b_{201} & b_{202} \\ b_{010} & b_{011} & b_{012} & b_{110} & b_{111} & b_{012} & b_{210} & b_{211} & b_{212} \\ b_{020} & b_{021} & b_{022} & b_{120} & b_{121} & b_{022} & b_{220} & b_{221} & b_{222} \end{array}\right\|,$$

$$C_{lsm}=\left\|\begin{array}{ccc|ccc|ccc} c_{000} & c_{001} & c_{002} & c_{010} & c_{011} & c_{012} & c_{020} & c_{021} & c_{022} \\ c_{100} & c_{101} & c_{102} & c_{110} & c_{111} & c_{112} & c_{120} & c_{121} & c_{122} \\ c_{200} & c_{201} & c_{202} & c_{210} & c_{211} & c_{212} & c_{220} & c_{221} & c_{222} \end{array}\right\|.$$

Тогда элементы (1, 1)-свернутого произведения $C={}^{(1,\,1)}A\times B$ с одним скоттовым индексом s и одним кэлиевым индексом c вычисляются по формулам, приведенным в таблице 3.1. Ни один элемент одного столбца таблицы не встречается в других столбцах, что соответствует утверждению 3.1.

Таблица 3.1. Вычисление 2-мерных сечений ориентации s матрицы

$C_{(s)}\|s=0$	$C_{(s)}\|s=1$	$C_{(s)}\|s=2$
$c_{000}=a_{000}b_{000}+a_{001}b_{010}+a_{002}b_{020}$	$c_{010}=a_{010}b_{100}+a_{011}b_{110}+a_{012}b_{120}$	$c_{020}=a_{020}b_{200}+a_{021}b_{210}+a_{022}b_{220}$
$c_{001}=a_{000}b_{001}+a_{001}b_{011}+a_{002}b_{021}$	$c_{011}=a_{010}b_{101}+a_{011}b_{111}+a_{012}b_{121}$	$c_{021}=a_{020}b_{201}+a_{021}b_{212}+a_{022}b_{221}$
$c_{002}=a_{000}b_{002}+a_{001}b_{012}+a_{002}b_{022}$	$c_{012}=a_{010}b_{102}+a_{011}b_{112}+a_{012}b_{122}$	$c_{022}=a_{020}b_{202}+a_{021}b_{212}+a_{022}b_{222}$
$c_{100}=a_{100}b_{000}+a_{101}b_{010}+a_{102}b_{020}$	$c_{110}=a_{110}b_{100}+a_{111}b_{110}+a_{112}b_{120}$	$c_{120}=a_{120}b_{200}+a_{121}b_{210}+a_{122}b_{220}$
$c_{101}=a_{100}b_{001}+a_{101}b_{011}+a_{102}b_{021}$	$c_{111}=a_{110}b_{101}+a_{111}b_{111}+a_{112}b_{121}$	$c_{121}=a_{120}b_{201}+a_{121}b_{212}+a_{122}b_{221}$

$c_{102}=a_{100}b_{002}+a_{001}b_{012}+a_{102}b_{022}$	$c_{112}=a_{110}b_{102}+a_{111}b_{112}+a_{112}b_{122}$	$c_{122}=a_{120}b_{202}+a_{121}b_{212}+a_{122}b_{222}$
$c_{200}=a_{200}b_{000}+a_{201}b_{010}+a_{202}b_{020}$ $c_{201}=a_{200}b_{001}+a_{201}b_{011}+a_{202}b_{021}$ $c_{202}=a_{200}b_{002}+a_{201}b_{012}+a_{202}b_{022}$	$c_{210}=a_{210}b_{100}+a_{211}b_{110}+a_{212}b_{120}$ $c_{211}=a_{210}b_{101}+a_{211}b_{111}+a_{212}b_{121}$ $c_{212}=a_{210}b_{102}+a_{211}b_{112}+a_{212}b_{122}$	$c_{220}=a_{220}b_{200}+a_{221}b_{210}+a_{222}b_{220}$ $c_{221}=a_{020}b_{201}+a_{021}b_{212}+a_{022}b_{221}$ $c_{222}=a_{220}b_{202}+a_{221}b_{212}+a_{222}b_{222}$

Таким образом (1, 1)-свернутое произведение трехмерных матриц сводится к n_s (в примере к трем) (0, 1)-свернутым произведениям ориентации s.

Очевидно, что в общем случае наибольшее число (0, μ)-свернутых произведений сечений ориентации $(s_1, ..., s_\lambda)$ матриц-сомножителей равно $n_{s_1} \times ... \times n_{s_\lambda}$.

Как и в случае умножения плоских матриц, алгоритм умножения многомерных матриц – простых сечений матриц A и B ориентации $(s_1, ..., s_\lambda)$ реализуется параллельно выполняемыми процессами. Поскольку умножения этих сечений не зависят друг от друга, далее рассматривается умножение в рамках только одного сечения. Каждый процесс выполняет умножение блоков этих матриц A и B, в результате которого получается блок матрицы C.

Сложность обобщения заключается в том, в алгоритме Кэннона, реализующем умножение плоских матриц, операции циклического сложения выполняются над тремя индексами, а в случае многомерных матриц необходимо выполнять эту операцию над $\kappa+\lambda+\nu$ индексами. Это существенно усложняет как формализацию задачи пересылки блоков матриц операндов между процессами, так и реализацию алгоритма пересчета значений индексов. Поскольку предполагается, что все индексы всех сечений принимают одно и то же количество значений от 0 до n-1, то далее рассматривается метод, основанный на представлении каждого набора значений индексов в виде ***числа в системе счисления с основанием E*** ($E<n$), смысл которого будет определен в дальнейшем.

Пусть T такое число, что n^κ, n^μ и n^ν кратны T и $E=\frac{n}{T}$. Тогда матрицы A и B могут быть разбиты на $E^{2(\kappa+\mu)}$ и $E^{2(\mu+\nu)}$ блоков соответственно. Следуя требова-

ниям алгоритма Кэннона, предполагается, что $\kappa=\nu$, то есть обе матрицы содержат одинаковое количество свободных индексов. Следовательно, в этом случае для параллельной реализации (0, μ)-свернутого произведения требуется $E^{2(\kappa+\mu)} = E^{2(\mu+\nu)}$ процессов. Каждый процесс последовательно выполняет умножение соответствующих друг другу блоков условно двумерных матриц A и B и складывает произведение с блоком условно двумерной матрицы C. После выполнения последней итерации блок матрицы C принимает окончательное значение. При этом должны выполняться следующие требования:

1. Блок матрицы A – это ее сечение, в котором индексы разбиения l и c принимают E последовательных значений, индексы разбиения s – единственное значение.
2. Блок матрицы B – это ее сечение, в котором индексы разбиения m и c принимают E последовательных значений, индексы разбиения s – единственное значение.
3. Блок матрицы C – это ее сечение, в котором индексы разбиения l и m принимают n последовательных значений, индексы разбиения s – единственное значение.

Общее число процессов, необходимых для вычисления одного сечения ориентации $(s_1, ..., s_\lambda)$ матрицы-результата C, определяется, общим числом индексов разбиений l и m, равным $\kappa + \nu$ и, в рассматриваемом случае, числом E, которому кратно n – количество значений каждого индекса. Тогда индекс процесса можно представить в виде триады: $N_E^l < s_1^0, ..., s_\lambda^0 > N_E^m$, где N_E^l и N_E^m – числа в E-ричной системе счисления, принимающие значения от 0 до E^κ–1 и E^μ–1. Следовательно, количество процессов, необходимых для вычисления одного сечения матрицы C равно $E^{\kappa+\nu}$. Тем самым, многомерные матрицы-операнды каждого (0, μ)-свернутого произведения можно рассматривать, как условно трехмерные. Далее приводится описание алгоритма блочного умножения многомерных матриц, аналогичного алгоритму Кэннона.

3.4.2. Описание алгоритма умножения многомерных матриц

Как и большинство алгоритмов умножения матриц, алгоритм умножения многомерных матриц [110, 170] состоит из двух этапов.

На первом этапе производится "обнуление" (заполнение значениями нейтрального элемента) блоков сечения матрицы *C* в соответствующих процессах и распределение по процессам блоков сечений матриц *A* и *B*. Распределение может быть физическим, если каждый процесс связан с автономным запоминающим устройством, или логическим, если все процессы обрабатывают данные, расположенные на одном запоминающем устройстве.

Поскольку, как показано в разделе 3.4.1, общий индекс процесса можно представить в виде триады: $< N_E^l s_1^*, ..., s_\lambda^* N_E^m >$, где N_E^l и N_E^m – числа в *E*-ричной системе счисления, принимающие значения от 0 до $E^\kappa - 1 = E^\nu - 1$ и соответствующие свободным индексам обеих матриц. Пересчет индексов процессов производится посредством сложения чисел N_E^l и N_E^m с целым числом, которое задается на каждом шаге алгоритма. Фиксированные значения $s_1^*, ..., s_\lambda^*$ скотговых индексов в пересчете не участвуют и используются только для идентификации сечения. Таким образом многомерные сечения условно трехмерных матриц сводятся к условно двумерным матрицам и (0, μ)-свернутое произведение сводится к условно (0, 1)-свернутому произведению, что позволяет упростить обобщение алгоритма Кэннона.

В начальном состоянии предполагается, что каждый процесс связан с блоками матриц *A* и *B*, индексы которых соответствуют индексам процесса. Это означает, что для сечения ориентации $s_1^*, ..., s_\lambda^*$ при фиксированных *i* и *j* $i = 0, ..., \underbrace{E-1...E-1}_{\kappa}$, $j = \underbrace{E-1...E-1}_{\nu}$, где $\underbrace{E-1...E-1}_{\kappa} = E^\kappa - 1$, $\underbrace{E-1...E-1}_{\nu} = E^\nu - 1$ процессу $P_{l_1^i ... l_\kappa^i s_1^*, ..., s_\lambda^* m_1^i ... m_\kappa^i}$ будут соответствовать блоки $A_{l_1^i ... l_\kappa^i s_1^*, ..., s_\lambda^* c_1^i ... c_\mu^i}$ и $B_{s_1^*, ..., s_\lambda^* c_1^i ... c_\mu^i m_1^i ... m_\nu^i}$ матриц *A* и *B*. Начальная привязка блоков матриц *A* и *B* к процессам приведена в таблице 3.2.

Таблица 3.2. Начальная привязка блоков матриц-операндов к процессам

$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*0\ldots0}$ $B_{s_1^*\ldots s_\lambda^*0\ldots00\ldots0}$	$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*0\ldots1}$ $B_{s_1^*\ldots s_\lambda^*0\ldots00\ldots1}$	$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots2}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*0\ldots2}$ $B_{s_1^*\ldots s_\lambda^*0\ldots00\ldots2}$	…	$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*0\ldots0\underbrace{E-1\ldots E-1E-2}_{\nu}}$	$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*0\ldots0\underbrace{E-1\ldots E-1}_{\nu}}$
$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\nu}}$ $A_{0\ldots1s_1^*\ldots s_\lambda^*0\ldots0}$ $B_{s_1^*\ldots s_\lambda^*0\ldots10\ldots0}$	$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\nu}}$ $A_{0\ldots1s_1^*\ldots s_\lambda^*0\ldots1}$ $B_{s_1^*\ldots s_\lambda^*0\ldots10\ldots1}$	$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots2}_{\nu}}$ $A_{0\ldots1s_1^*\ldots s_\lambda^*0\ldots2}$ $B_{s_1^*\ldots s_\lambda^*0\ldots10\ldots2}$	…	$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}$ $A_{0\ldots1s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*0\ldots1\underbrace{E-1\ldots E-1E-2}_{\nu}}$	$P_{001s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}$ $A_{001s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*001\underbrace{E-1\ldots E-1}_{\nu}}$
$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*0\ldots0}$ $B_{s_1^*\ldots s_\lambda^*0\ldots20\ldots0}$	$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*0\ldots1}$ $B_{s_1^*\ldots s_\lambda^*0\ldots20\ldots1}$	$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots2}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*0\ldots2}$ $B_{s_1^*\ldots s_\lambda^*0\ldots20\ldots2}$	…	$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*0\ldots2\underbrace{E-1\ldots E-1E-2}_{\nu}}$	$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*0\ldots2\underbrace{E-1\ldots E-1}_{\nu}}$
…					
$P_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*0\ldots0}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*0\ldots0}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-20\ldots0}_{\mu}\,\underbrace{}_{\nu}}$	$P_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*0\ldots1}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*0\ldots1}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-20\ldots1}_{\mu}\,\underbrace{}_{\nu}}$	$P_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*0\ldots2}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*0\ldots2}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-20\ldots2}_{\mu}\,\underbrace{}_{\nu}}$	…	$P_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}\underbrace{E-1\ldots E-1E-2}_{\nu}}$	$P_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}\underbrace{E-1\ldots E-1}_{\nu}}$
$P_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*0\ldots0}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*0\ldots0}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-10\ldots0}_{\mu}\,\underbrace{}_{\nu}}$	$P_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*0\ldots1}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*0\ldots1}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-10\ldots1}_{\mu}\,\underbrace{}_{\nu}}$	$P_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*0\ldots2}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*0\ldots2}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-10\ldots2}_{\mu}\,\underbrace{}_{\nu}}$	…	$P_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}\underbrace{E-1\ldots E-1E-2}_{\nu}}$	$P_{\underbrace{E-1\ldots E-1s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}_{\kappa}\,\underbrace{}_{\nu}}$ $A_{\underbrace{E-1\ldots E-1E-2s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}_{\kappa}\,\underbrace{}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}\underbrace{E-1\ldots E-1}_{\nu}}$

Таблица 3.3. Привязка блоков матриц-операндов к процессам после выполнения первого этапа алгоритма

$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*0\ldots0}$ $B_{s_1^*\ldots s_\lambda^*0\ldots00\ldots0}$	$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*0\ldots1}$ $B_{s_1^*\ldots s_\lambda^*0\ldots10\ldots1}$	$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots2}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*0\ldots2}$ $B_{s_1^*\ldots s_\lambda^*0\ldots20\ldots2}$	…	$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}\underbrace{E-1\ldots E-1E-2}_{\nu}}$	$P_{\underbrace{0\ldots0}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}$ $A_{0\ldots0s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}\underbrace{E-1\ldots E-1}_{\nu}}$
$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\nu}}$ $A_{0\ldots1s_1^*\ldots s_\lambda^*0\ldots1}$ $B_{s_1^*\ldots s_\lambda^*0\ldots10\ldots0}$	$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\nu}}$ $A_{0\ldots1s_1^*\ldots s_\lambda^*0\ldots2}$ $B_{s_1^*\ldots s_\lambda^*0\ldots20\ldots1}$	$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots2}_{\nu}}$ $A_{0\ldots1s_1^*\ldots s_\lambda^*0\ldots3}$ $B_{s_1^*\ldots s_\lambda^*0\ldots30\ldots2}$	…	$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}$ $A_{001s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}}$ $B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}\underbrace{E-1\ldots E-1E-2}_{\nu}}$	$P_{\underbrace{0\ldots1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}$ $A_{0\ldots1s_1^*\ldots s_\lambda^*0\ldots0}$ $B_{s_1^*\ldots s_\lambda^*0\ldots0\underbrace{E-1\ldots E-1}_{\nu}}$
$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*0\ldots2}$ $B_{s_1^*\ldots s_\lambda^*0\ldots20\ldots0}$	$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*0\ldots3}$ $B_{s_1^*\ldots s_\lambda^*0\ldots30\ldots1}$	$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots2}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*0\ldots4}$ $B_{s_1^*\ldots s_\lambda^*0\ldots40\ldots2}$	…	$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1E-2}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*0\ldots0}$ $B_{s_1^*\ldots s_\lambda^*0\ldots0\underbrace{E-1\ldots E-1E-2}_{\nu}}$	$P_{\underbrace{0\ldots2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\nu}}$ $A_{0\ldots2s_1^*\ldots s_\lambda^*0\ldots1}$ $B_{s_1^*\ldots s_\lambda^*0\ldots1\underbrace{E-1\ldots E-1}_{\nu}}$
…					

$P_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...0}_{\nu}}$ $A_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1E-2}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-20...0}_{\mu\quad\nu}}$	$P_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...1}_{\nu}}$ $A_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{E-1...E-10...1}_{\mu\quad\nu}}$	$P_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...2}_{\nu}}$ $A_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...0}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{0...00...2}_{\mu\quad\nu}}$	...	$P_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1E-2}_{\nu}}$ $A_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1E-4}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-4}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$	$P_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1}_{\nu}}$ $A_{\underbrace{E-1...E-1E-2}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1E-3}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}\underbrace{E-1...E-1}_{\nu}}$
$P_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...0}_{\nu}}$ $A_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{E-1...E-10...0}_{\mu\quad\nu}}$	$P_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...1}_{\nu}}$ $A_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...0}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{0...00...1}_{\mu\quad\nu}}$	$P_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...2}_{\nu}}$ $A_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*0...1}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{0...10...2}_{\mu\quad\nu}}$	...	$P_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1E-2}_{\nu}}$ $A_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1E-3}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-3}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$	$P_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1}_{\nu}}$ $A_{\underbrace{E-1...E-1}_{\kappa}\underbrace{s_1^*...s_\lambda^*}\underbrace{E-1...E-1E-2}_{\mu}}$ $B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}\underbrace{E-1...E-1}_{\nu}}$

Распределение блоков основано на том, что наборы фиксированных значений индексов разбиений l и m можно рассматривать как числа в E-ричной системе счисления $(l_1^i...l_\kappa^i)_E$ и $(m_1^j...m_\nu^j)_E$. Тогда первый этап алгоритма состоит из следующих шагов:

На первом этапе алгоритма выполняются следующие действия:

1. Блок матрицы A с набором значений индексов $(l_1^i...l_\kappa^i)_E$ разбиения l связывается с процессом, набор значений индексов разбиения m которого вычисляется по формуле. $(E^\kappa + (l_1^i...l_\kappa^i)_E - i) \bmod E^\kappa\ (i = 0, ..., \underbrace{E-1...E-1}_{\kappa})$.

2. Блок матрицы B с набором значений индексов $(m_1^j...m_\nu^j)_E$ разбиения m связывается с процессом, набор значений индексов разбиения m которого вычисляется по формуле. $(E^\nu + (m_1^j...m_\nu^j)_E - j) \bmod E^\nu\ (j = 0, ..., \underbrace{E-1...E-1}_{\nu})$.

Привязка блоков матриц A и B к процессам после завершения первого этапа алгоритма приведена в таблице 3.3.

Второй этап алгоритма состоит из E итераций, на каждой из которых выполняются три действия:

1. Каждый процесс $P_{l_1^i...l_\kappa^i s_1^*, ..., s_\lambda^* m_1^i...m_\kappa^i}$ выполняет умножение связанных с ним блоков матриц A и B и складывает результат умножения с соответствующим ему блоком матрицы C.

2. Блок матрицы A с набором значений индексов $(l_1^i...l_\kappa^i)_E$ разбиения l связывается с процессом, набор значений индексов разбиения m которого вычисляется по формуле $(E^\kappa + (l_1^i...l_\kappa^i)_E - 1) \bmod E^\kappa$ $(i = 0, ..., \underbrace{E-1...E-1}_{\kappa})$..

3. Блок матрицы B с набором значений индексов $(m_1^j...m_\nu^j)_E$ разбиения m связывается с процессом, набор значений индексов разбиения m которого вычисляется по формуле. $(E^\nu + (m_1^j...m_\nu^j)_E - 1) \bmod E^\nu$ $(j = 0, ..., \underbrace{E-1...E-1}_{\nu})$.

Последовательности произведений блоков матриц A и B, участвующих вычислениях блоков матрицы C на всех E итерациях приведены в таблице 3.4.

Таблица 3.4. Выполнение действий на итерациях второго этапа алгоритма

Индекс процесса	Итерация	Слагаемое (произведение блоков матриц A и B)
$\underset{\kappa}{0...0}\,\underset{\nu}{0...0}$	0	$A_{\underset{\kappa}{0..0}s_1^*...s_\lambda^*\underset{\mu}{0...0}} \times B_{s_1^*...s_\lambda^*\underset{\mu}{0...0}\underset{\nu}{0...0}}$
	1	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underset{\mu}{0...1}} \times B_{s_1^*...s_\lambda^*\underset{\mu}{0...1}\underset{\nu}{0...0}}$
	…	
	$E-2$	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}\underset{\nu}{0...0}}$
	$E-1$	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-1}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-1}_{\mu}\underset{\nu}{0...0}}$
$\underset{\kappa}{0...0}\,\underset{\nu}{0...1}$	0	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underset{\mu}{0...1}} \times B_{s_1^*...s_\lambda^*\underset{\mu}{0...1}\underset{\nu}{0...1}}$
	1	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underset{\mu}{0...2}} \times B_{s_1^*...s_\lambda^*\underset{\mu}{0...2}\underset{\nu}{0...1}}$
	…	
	$E-2$	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underbrace{E-1...E-1}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1}_{\mu}\underset{\nu}{0...1}}$
	$E-1$	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underset{\mu}{0...0}} \times B_{s_1^*...s_\lambda^*\underset{\mu}{0...0}\underset{\nu}{0...1}}$
…		
$\underset{\kappa}{0...0}\,\underbrace{E-1...E-1E-2}_{\nu}$	0	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$
	1	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underbrace{E-1...E-1}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$
	…	
	$E-2$	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-4}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-4}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$
	$E-1$	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-3}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-3}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$
$\underset{\kappa}{0...0}\,\underbrace{E-1...E-1}_{\nu}$	0	$A_{\underset{\kappa}{0...0}s_1^*...s_\lambda^*\underbrace{E-1...E-1}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1}_{\mu}\underbrace{E-1...E-1}_{\nu}}$
	1	$A_{\underbrace{0...0}_{\kappa}s_1^*...s_\lambda^*\underbrace{0...0}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{0...0}_{\mu}\underbrace{E-1...E-1}_{\nu}}$
	…	

	$E-2$	$A_{\underbrace{0\ldots0}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-3}_{\mu}\underbrace{E-1\ldots E-1\,E-2}_{\nu}}$
	$E-1$	$A_{\underbrace{0\ldots0}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}\underbrace{E-1\ldots E-1\,E-2}_{\nu}}$
$\underset{\kappa}{0\ldots1}\underset{\nu}{0\ldots0}$	0	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{0\ldots1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{0\ldots1}_{\mu}\underbrace{0\ldots0}_{\nu}}$
	1	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-3}_{\mu}\underbrace{0\ldots0}_{\nu}}$
	…	
	$E-2$	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}\underbrace{0\ldots0}_{\nu}}$
	$E-1$	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{0\ldots0}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{0\ldots0}_{\mu}\underbrace{0\ldots0}_{\nu}}$
$\underset{\kappa}{0\ldots1}\underset{\nu}{0\ldots1}$	0	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{0\ldots2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{0\ldots2}_{\mu}\underbrace{0\ldots1}_{\nu}}$
	1	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{0\ldots3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{0\ldots3}_{\mu}\underbrace{0\ldots1}_{\nu}}$
	…	
	$E-2$	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}\underbrace{0\ldots1}_{\nu}}$
	$E-1$	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{0\ldots1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{0\ldots1}_{\mu}\underbrace{0\ldots1}_{\nu}}$
…		
$\underset{\kappa}{0\ldots1}\underbrace{E-1\ldots E-1E-2}_{\nu}$	0	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}\underbrace{E-1\ldots E-1\,E-2}_{\nu}}$
	1	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{0\ldots0}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{0\ldots0}_{\mu}\underbrace{E-1\ldots E-1\,E-2}_{\nu}}$
	…	
	$E-2$	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-3}_{\mu}\underbrace{E-1\ldots E-1E-2}_{\nu}}$
	$E-1$	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}\underbrace{E-1\ldots E-1E-2}_{\nu}}$
$\underset{\kappa}{0\ldots1}\underbrace{E-1\ldots E-1}_{\nu}$	0	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{0\ldots0}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{0\ldots0}_{\mu}\underbrace{E-1\ldots E-1}_{\nu}}$
	1	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{0\ldots1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{0\ldots1}_{\mu}\underbrace{E-1\ldots E-1}_{\nu}}$
	…	
	$E-2$	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-2}_{\mu}\underbrace{E-1\ldots E-1}_{\nu}}$
	$E-1$	$A_{\underbrace{0\ldots1}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1E-1}_{\mu}\underbrace{E-1\ldots E-1}_{\nu}}$
…		
$\underbrace{E-1\ldots E-1E-2}_{\kappa}\underset{\nu}{0\ldots0}$	0	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1\,E-2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1\,E-2}_{\mu}\underbrace{0\ldots0}_{\nu}}$
	1	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1}_{\mu}\underbrace{0\ldots0}_{\nu}}$
	…	
	$E-2$	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1\,E-4}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\underbrace{E-1\ldots E-1\,E-4}_{\mu}\underbrace{0\ldots0}_{\nu}}$

	$E-1$	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3\,0\ldots0}_{\mu}\,\underbrace{0}_{\nu}}$
$\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,\underbrace{0\ldots1}_{\nu}$	0	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\mu}\,\underbrace{0\ldots1}_{\nu}}$
	1	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\mu}\,\underbrace{0\ldots1}_{\nu}}$
		…
	$E-2$	$A_{\underbrace{E-1\ldots E-1\,E-3}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3\,0\ldots1}_{\mu}\,\underbrace{}_{\nu}}$
	$E-1$	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-2\,0\ldots1}_{\mu}\,\underbrace{}_{\nu}}$
		…
$\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,\underbrace{E-1\ldots E-1\,E-2}_{\nu}$	0	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-4}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-4}_{\mu}\,\underbrace{E-1\ldots E-1\,E-2}_{\nu}}$
	1	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3}_{\mu}\,\underbrace{E-1\ldots E-1\,E-2}_{\nu}}$
		…
	$E-2$	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-6}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-6}_{\mu}\,\underbrace{E-1\ldots E-1\,E-2}_{\nu}}$
	$E-1$	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-5}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-5}_{\mu}\,\underbrace{E-1\ldots E-1\,E-2}_{\nu}}$
$\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,\underbrace{E-1\ldots E-1}_{\nu}$	0	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3}_{\mu}\,\underbrace{E-1\ldots E-1}_{\nu}}$
	1	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-2}_{\mu}\,\underbrace{E-1\ldots E-1}_{\nu}}$
		…
	$E-2$	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-5}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-5}_{\mu}\,\underbrace{E-1\ldots E-1}_{\nu}}$
	$E-1$	$A_{\underbrace{E-1\ldots E-1\,E-2}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-4}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-4}_{\mu}\,\underbrace{E-1\ldots E-1}_{\nu}}$
$\underbrace{E-1\ldots E-1}_{\kappa}\,\underbrace{0\ldots0}_{\nu}$	0	$A_{\underbrace{E-1\ldots E-1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-1\,0\ldots0}_{\mu}\,\underbrace{}_{\nu}}$
	1	$A_{\underbrace{E-1\ldots E-1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\mu}\,\underbrace{0\ldots0}_{\nu}}$
		…
	$E-2$	$A_{\underbrace{E-1\ldots E-1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-3\,0\ldots0}_{\mu}\,\underbrace{}_{\nu}}$
	$E-1$	$A_{\underbrace{E-1\ldots E-1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-2\,0\ldots0}_{\mu}\,\underbrace{}_{\nu}}$
$\underbrace{E-1\ldots E-1}_{\kappa}\,\underbrace{0\ldots1}_{\nu}$	0	$A_{\underbrace{E-1\ldots E-1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots0}_{\mu}\,\underbrace{0\ldots1}_{\nu}}$
	1	$A_{\underbrace{E-1\ldots E-1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{0\ldots1}_{\mu}\,\underbrace{0\ldots1}_{\nu}}$
		…
	$E-2$	$A_{\underbrace{E-1\ldots E-1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-2}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,E-2\,0\ldots1}_{\mu}\,\underbrace{}_{\nu}}$
	$E-1$	$A_{\underbrace{E-1\ldots E-1}_{\kappa}\,s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1}_{\mu}} \times B_{s_1^*\ldots s_\lambda^*\,\underbrace{E-1\ldots E-1\,0\ldots1}_{\mu}\,\underbrace{}_{\nu}}$

…		
$\underbrace{E-1...E-1}_{\kappa}\underbrace{E-1...E-1E-2}_{\nu}$	0	$A_{\underbrace{E-1...E-1}_{\kappa}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-3}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-3}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$
	1	$A_{\underbrace{E-1...E-1}_{\kappa}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$
	…	
	$E-2$	$A_{\underbrace{E-1...E-1}_{\kappa}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-5}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-5}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$
	$E-1$	$A_{\underbrace{E-1...E-1}_{\kappa}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-4}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-4}_{\mu}\underbrace{E-1...E-1E-2}_{\nu}}$
$\underbrace{E-1...E-1}_{\kappa}\underbrace{E-1...E-1}_{\nu}$	0	$A_{\underbrace{E-1...E-1}_{\kappa}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-2}_{\mu}\underbrace{E-1...E-1}_{\nu}}$
	1	$A_{\underbrace{E-1...E-1}_{\kappa}s_1^*...s_\lambda^*\underbrace{E-1...E-1}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1}_{\mu}\underbrace{E-1...E-1}_{\nu}}$
	…	
	$E-2$	$A_{\underbrace{E-1...E-1}_{\kappa}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-4}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-4}_{\mu}\underbrace{E-1...E-1}_{\nu}}$
	$E-1$	$A_{\underbrace{E-1...E-1}_{\kappa}s_1^*...s_\lambda^*\underbrace{E-1...E-1E-3}_{\mu}} \times B_{s_1^*...s_\lambda^*\underbrace{E-1...E-1E-3}_{\mu}\underbrace{E-1...E-1}_{\nu}}$

Пример 3.5. В этом примере рассматривается реализация алгоритма умножения многомерных матриц при условиях подобных рассмотренным в примере 3.4. Пусть E=3, тогда $A=\|A_{lsc}\|$, $B=\|B_{scm}\|$ и $C=\|C_{lsm}\|$ – трехмерные матрицы, индексы которых принимают значения от 0 до 2, а элементы – блоки матриц *A*, *B* и *C*. Распределение блоков произвольного сечения ориентации s матриц *A*, *B* и C по процессам после выполнения первого этапа алгоритма показано в таблице 3.5.

Таблица 3.5. Распределение блоков сечения матриц A, B и C по процессам

P_{0s0}	C_{0s0}, A_{0s0}, B_{s00}	P_{1s0}	C_{1s0}, A_{1s0}, B_{s00}	P_{2s0}	C_{2s0}, A_{2s2}, B_{s20}
P_{0s1}	C_{0s1}, A_{0s1}, B_{s11}	P_{1s1}	C_{1s1}, A_{1s1}, B_{s11}	P_{2s1}	C_{2s1}, A_{2s0}, B_{s20}
P_{0s2}	C_{0s1}, A_{0s2}, B_{s22}	P_{1s2}	C_{1s2}, A_{1s0}, B_{s02}	P_{2s2}	C_{2s2}, A_{2s1}, B_{s12}

Схема перемещения блоков сечений матриц *A* и *B* между процессами на втором этапе алгоритма при $s=0$ показана на рисунке 3.4.

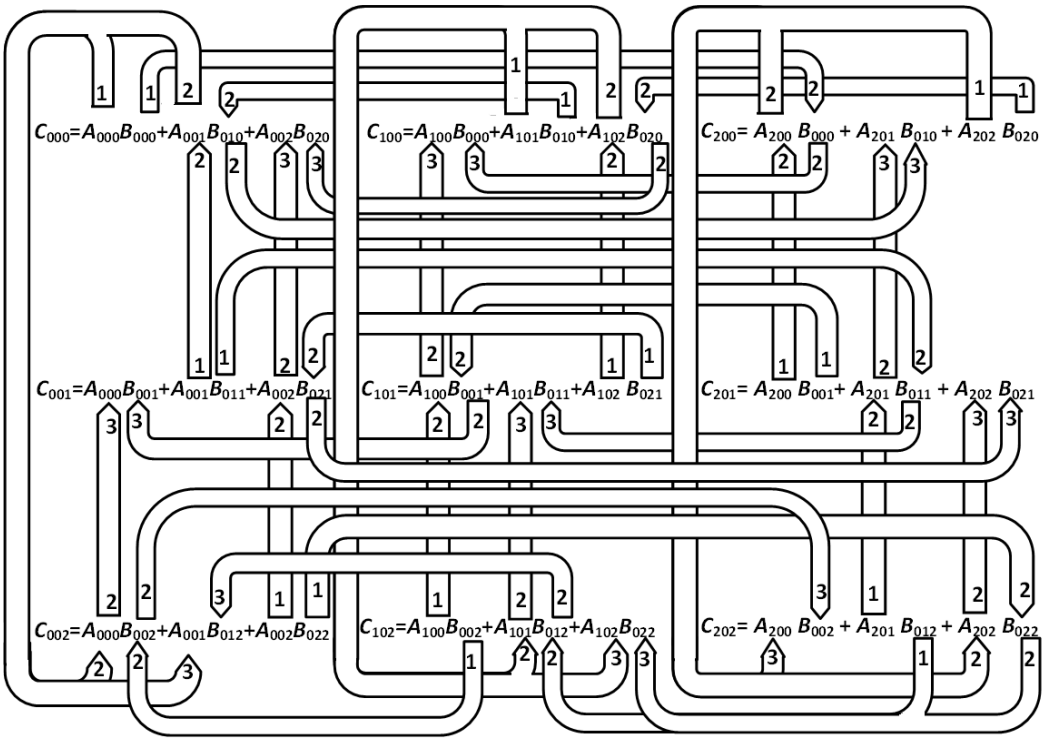


Рис. 3.4. Схема пересылки блоков сечения на втором этапе алгоритма

Последовательности произведений блоков сечений матриц A и B в каждом процессе на втором этапе алгоритма при $s=0$ приведены в таблице 3.6.

Таблица 3.6. Порядок вычисления блоков сечения матрицы C

Процесс	Слагаемые		
	итерация 1	итерация 2	итерация 3
P_{000}	$A_{000}B_{000}$	$A_{001}B_{010}$	$A_{002}B_{020}$
P_{001}	$A_{001}B_{011}$	$A_{002}B_{021}$	$A_{000}B_{001}$
P_{002}	$A_{002}B_{022}$	$A_{000}B_{002}$	$A_{001}B_{012}$
P_{100}	$A_{101}B_{010}$	$A_{102}B_{020}$	$A_{100}B_{000}$
P_{101}	$A_{102}B_{021}$	$A_{100}B_{001}$	$A_{101}B_{011}$
P_{102}	$A_{100}B_{002}$	$A_{001}B_{012}$	$A_{102}B_{022}$
P_{200}	$A_{202}B_{020}$	$A_{200}B_{000}$	$A_{201}B_{010}$
P_{201}	$A_{200}B_{001}$	$A_{201}B_{011}$	$A_{202}B_{021}$
P_{202}	$A_{201}B_{012}$	$A_{202}B_{022}$	$A_{200}B_{002}$

Рассмотренный алгоритм умножения многомерных матриц обладает одним важным качеством, которое следует из утверждения 3.1. Независимость умножений сечений матриц-сомножителей по скоттовым индексам (ориента-

ции $(s_1, ..., s_\lambda)$) порождает дополнительный уровень параллелизма, на котором гиперпроцессы $\mathbb{P}_1$, …, $\mathbb{P}_s$ (наибольшее значение $s = n_{s_1} \times ... \times n_{s_\lambda}$, если все сечения простые) параллельно реализуют параллельные алгоритмы умножения сечений матриц *A* и *B*. Эти сечения имеют меньшую размерность или меньшее количество элементов чем матрицы *A* и *B*.

3.5. Алгоритм параллельной реализации операции слияния нестрого упорядоченных файлов

3.5.1. Анализ алгоритмов параллельной реализации операции слияния нестрого упорядоченных файлов

Как было показано в главе 2, операция слияния нестрого упорядоченных файлов соответствует операции (λ, μ)-свернутого произведения многомерных матриц, операции Join в реляционной модели SQL и аналогичным ей операциям в других моделях данных. Это, во всех смыслах, наиболее сложная операция обработки данных. С одной стороны, алгоритмы ее реализации, независимо от того реализуется она последовательно или параллельно, сложны в разработке, с другой – они имеют большую вычислительную сложность и требуют много оперативной памяти и машинного времени. Поэтому уже длительное время ведутся исследования в области анализа и разработки эффективных параллельных алгоритмов реализации операции Join [178, 179]. В настоящее время ведутся разработки с использованием современных средств вычислительной техники [180, 181], в том числе и в ставшем популярным направлении обработки данных в оперативной памяти (In-Memory Database) [182, 183]. Далее рассматривается параллельный алгоритм реализации операции слияния нестрого упорядоченных файлов, основанный на предложенном способе распределения данных.

Согласно определению, данному в главе 2, в операции слияния нестрого упорядоченных файлов участвуют два файла X_L и Y_M. При реализации операции множество ключей файла-результата Z_K формируется на основе множеств клю-

чей файлов-операндов X_L и Y_M. Добавление новых ключей в файл-результат не влияет на алгоритм, реализующий слияние нестрого упорядоченных файлов. Поэтому, без ограничения общности, при разработке алгоритма можно считать, что все три файла: X_K, Y_K и Z_K имеют одно и то же множество ключей K. Если $\{K^*\}_X$ – совокупность всех экземпляров множества ключей K в записях файла X_K, а $\{K^*\}_Y$ – совокупность всех экземпляров множества ключей K в записях файла Y_K, то, в общем случае, совокупность всех экземпляров множества ключей K в записях файла Z_K определяется как $\{K^*\}_Z = \{K^*\}_X \cap \{K^*\}_Y$. Тогда можно определить файл-результат Z_K как $Z_K = \bigcup_{i=1}^{n} f(X_{K^*_{(i)}} \times Y_{K^*_{(i)}})$, n – количество элементов в множестве $\{K^*\}_Z$. Это определяет свойства алгоритмов реализации операции слияния нестрого упорядоченных файлов.

Для того чтобы было понятно каким должен быть параллельный алгоритм, реализующий операцию слияния нестрого упорядоченных файлов, далее рассматриваются последовательные алгоритмы ее реализующие. В логически-последовательной обработке данных используются два основных алгоритма.

Алгоритм 1. Предполагается, что оба файла-операнда упорядочены по множеству ключей K. Один из файлов, пусть X_L, определяется как ведущий, другой, Y_M, – как ведомый.

Считывание ведущего файла в оперативную память осуществляется последовательно классами эквивалентности X_{K^*}. Для каждого прочитанного класса эквивалентности X_{K^*} продолжается последовательное считывание ведомого файла Y_M, и если ведомый файл содержит класс эквивалентности Y_{K^*}, то каждая запись этого класса эквивалентности обрабатывается со всеми расположенными в оперативной памяти записями класса эквивалентности X_{K^*}. В результате этой обработки формируются записи класса эквивалентности Z_{K^*} файла-результата. Процесс считывания и обработки продолжается до тех пор, пока один из файлов не будет прочитан полностью.

Алгоритм 2. Предполагается, что оба файла-операнда по-разному упорядочены по множеству ключей K. Ведущий файл X_L упорядочивается по множеству ключей L при условии, что ключи множества ключей K младшие (при сравнении следуют за остальными ключам из L). Ведомый файл Y_M упорядочивается по множеству ключей K, то есть превращается в файл Y_K.

Класс эквивалентности файла X_L считывается последовательно по одной записи. Прочитанная запись сравнивается по множеству ключей K с очередной записью файла Y_K, и, при выполнении условия слияния, обе записи обрабатываются и включаются в формирование записи файла-результата. После завершения считывания очередного класса эквивалентности ведущего файла X_L, ведомый файл Y_K начинает считываться с начала (открывается заново). Процесс считывания и обработки продолжается до тех пор, пока ведущий файл не будет прочитан полностью.

Очевидно, что первый алгоритм требует больших объемов оперативной памяти для размещения в ней целого класса эквивалентности ведущего файла, а второй – значительного машинного времени для многократно считывания с начала ведомого файла. В современных СУБД используются различные средства оптимизации рассматриваемой операции (Join), например, хеширование. Кроме того, в современных системах разработчик запроса имеет возможность указать исполнителю (компилятору) запроса какой алгоритм слияния предпочтителен.

3.5.2. Организация данных для параллельной реализации операции слияния нестрого упорядоченных файлов

Для параллельной реализации рассматриваемой операции необходимо, чтобы файлы-операнды были распределены между процессами так, как это показано на рисунке 3.5. Как и в случае алгоритма умножения многомерных матриц распределение может быть физическим или логическим. И в том и в другом случае способ распределения зависит от архитектуры вычислительного комплекса.

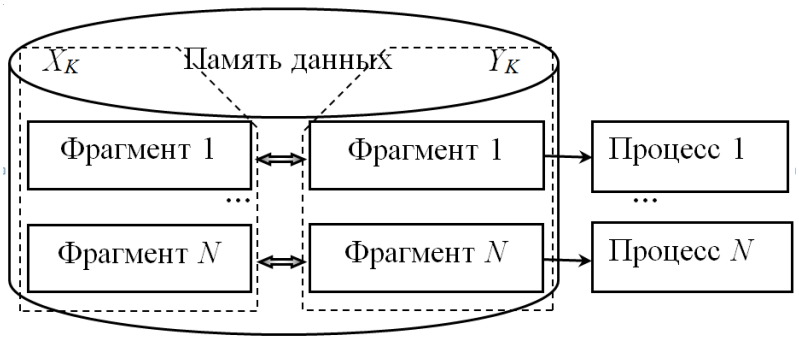


Рис. 3.5. Распределение файлов-операндов между процессами

Это означает, что если операция реализуется посредством N параллельных процессов, то файлы X_K и Y_K должны быть разделены на N фрагментов, которые удовлетворяют следующим требованиям:

1. Фрагменты файлов X_K и Y_K с номером i (i=1, …, N) связаны с процессом P_i. Они содержат классы эквивалентности с одинаковыми значениями экземпляров множества ключей K. То есть, если класс эквивалентности, соответствующий экземпляру множества ключей K^*, содержится в i-том фрагменте файла X_K, то аналогичный класс эквивалентности содержится и в i-том фрагменте файла Y_K. Или, для любого i из $X_{K^*_{(i)}} \neq \Theta$ следует $Y_{K^*_{(i)}} \neq \Theta$.

2. Каждый класс эквивалентности полностью расположен в одном фрагменте соответствующего файла. То есть все записи, принадлежащие одному классу эквивалентности, располагаются только в одном фрагменте соответствующего файла операнда.

3. Каждый процесс формирует свой фрагмент файла-результата Z_K. Эти фрагменты не пересекаются, а сам файл-результат формируется как их объединение.

Реализация распределения классов эквивалентности файлов X_K и Y_K по фрагментам осуществляется на основе системных метаданных, которые обеспечивают индексно-последовательное представление данных из этих файлов. Это позволяет использовать эффективный в рассматриваемом случае индексно-последовательный метод доступа (ISAM) [184, 185]. При использовании ISAM данные хранятся в последовательных (или логически последовательных) в файлах X_K и Y_K и упорядочены по множеству ключей K. Каждому файлу данных

ставится в соответствие файл (хеш-таблица), содержащий метаданные (индексы), служащие для доступа к записям каждого класса эквивалентности [117].

Для реализации распределения классов эквивалентности файлов X_K и Y_K, им ставятся в соответствие индексные файлы:

$indX_K = (< K^*_{(1)}, I_{1X}, m_{1X} >, ..., < K^*_{(n)}, I_{nX}, m_{nX} >)$ и $indY_K = (< K^*_{(1)}, I_{1Y}, m_{1Y} >, ..., < K^*_{(n)}, I_{nY}, m_{nY} >)$

(n – общее количество экземпляров множества ключей K).

Здесь $K^*_{(j)}$ – экземпляр множества ключей K, I_{jX} (I_{jY}) – индекс первой записи класса эквивалентности $X_{K^*_{(j)}}$ ($Y_{K^*_{(j)}}$), m_{jX} (m_{jY}) – количество записей в классе эквивалентности, соответствующем этому экземпляру множества ключей. В обоих индексных файлах, если класс эквивалентности $X_{K^*_{(j)}}$ состоит из единственной универсальной неопределенной записи Θ, то значение I_{jX} (I_{jY}) не определено, а m_{jX} (m_{jY})=0. Естественно, реализации файлов $indX_K$ и $indY_K$ не содержат записи с неопределенными значениями. Построение индексных файлов $indX_K$ и $indY_K$ осуществляется однопроходными алгоритмами, не требующими дополнительной оперативной памяти. Для реализации операции важно и то, что оба индексных файла строго упорядочены. Кроме того, на практике, в большинстве случаев, размер записи файла данных значительно больше размера записи соответствующего ему индексного файла.

Очевидно, что файл-результат будет содержать только те классы эквивалентности, экземпляры множества ключей которых принадлежат пересечению индексных файлов операндов ($indX_K \cap indY_K$). Из этого следует, что перед выполнением операции слияния нестрого упорядоченных файлов целесообразно выполнить операцию слияния соответствующих им строго упорядоченных индексных файлов, реализующую операцию пересечения. Тогда объемы обрабатываемых данных уменьшатся, а время выполнения операции слияния нестрого упорядоченных файлов сократится.

В самом простом случае распределения файлов по процессам каждый фрагмент файла-операнда содержит единственный класс эквивалентности. То-

гда время выполнения операции слияния нестрого упорядоченных файлов X_K и Y_K будет равно времени выполнения самого сложного декартова произведения классов эквивалентности. Если предположить, что оба класса эквивалентности $X_{K^*_{(j)}}$ и $Y_{K^*_{(j)}}$ расположены в оперативной памяти, то это время будет пропорционально величине $m_{jX} \times m_{jY}$. В реальных условиях фрагменты могут содержать несколько классов эквивалентности. Тогда процесс, реализующий обработку пары, фрагменты которой содержат по p классов эквивалентности файлов X_K и Y_K, формирует $R = \sum_{j=1}^{p} m_{jX} \times m_{jY}$ записей файла-результата Z_K. Значения R для разных процессов будут различными. Распараллеливание будет тем эффективнее, чем будет меньше значение разности $R_{\max}$ и $R_{\min}$.

Предложенный принцип распределения данных можно определить, как принцип симметричного горизонтального распределения данных. Действительно:

– файлы-операнды разбиваются на фрагменты так, как это принято при горизонтальном распределении данных;

– фрагменты файлов-операндов, связанные с одним процессом, содержат классы эквивалентности, соответствующие одним и тем же экземплярам множества ключей, то есть симметричны.

3.5.3. Параллельный алгоритм реализации операции слияния нестрого упорядоченных файлов

Так же, как и алгоритм умножения многомерных матриц, алгоритм операции слияния нестрого упорядоченных файлов состоит из двух этапов.

На первом этапе производится распределение фрагментов файлов-операндов между процессами. Отличие от распределения блоков матриц заключается в том, что классы эквивалентности содержат разное количество записей, в то время как все блоки матриц одинаковые. Поэтому для распределения фрагментов файлов необходимо использовать специальные оптимизационные методы, подобные методу загрузки рюкзака, например, метод, известный

как 0-1 мультипликативный рюкзак. Недостаток этих методов состоит в том, что реализующие их алгоритмы имеют экспоненциальную вычислительную сложность. Некоторые авторы [186, 187] относят эти алгоритмы к классу NP. Для упрощения работы и сокращения времени распределения файлов-операндов предлагается эвристический алгоритм, описание которого приводится далее.

3.5.3.1. Эвристический алгоритм распределения

В этом разделе дано формальное описание задачи оптимального распределения, даны необходимые обозначения, представлен алгоритм ее решения и его анализ [188-194].

Пусть нам дан набор из n элементов, каждый из которых имеет вес w_j (j = 1, ..., n), и набор из m хранилищ, каждое из которых способно вместить необходимый объем данных. Значение переменной $x_{ij} \in \{0, 1\}$ $x_{ij}=1$ означает, что элемент j размещен в хранилище i (i=1, …, m; j=1, …, n), в противном случае $x_{ij}=0$. Кроме того должно выполняться условие: если $x_{ij} = x_{kj}$ то $i=k$, которое означает, что один объект может располагаться только в одном хранилище. Тогда целевая в общем виде функция имеет вид:

$$\text{Minimize } z = Max(\sum_{j=1}^{n} w_j x_{1j}, ..., \sum_{j=1}^{n} w_j x_{mj}) - Min(\sum_{j=1}^{n} w_j x_{1j}, ..., \sum_{j=1}^{n} w_j x_{mj}).$$

Эта целевая функция позволяет решить задачу симметричного горизонтального распределения файлов.

Для решения задачи оптимального распределения используется алгоритм, показанный на листинге 3.1 (язык программирования C#). На вход алгоритма подаются два массива:

– Упорядоченный по возрастанию или убыванию массив записей, которые содержат сведения об n объектах. Эти записи содержат по крайней мере два поля, среди которых есть два обязательных: идентификатор и вес объекта. Массив упорядочивается по весу объекта. В случае задачи симметричного горизонтального распределения файлов эти поля содержат ключ класса эквивалентности и

вычислительную сложность декартова произведения соответствующих этому ключу классов эквивалентности файлов-операндов.

– Массив коллекций объектов для каждого хранилища. Записи каждой коллекции также содержат идентификатор и вес item. Размерность этого массива m, а каждой коллекции не более $\left\lceil \frac{n}{m} \right\rceil$.

Кроме того, алгоритм получает количество объектов – n и хранилищ – m.

Листинг 3.1. UDD algorithm

```
public void UDD((<datatype>[] ItemRecords, ArrayList[] WarehouseRecords, int n,
int m)
{
 1.    int i, j;
 2.    j = 0;
 3.    for (i = 0; i < n / 2; i++)
 4.    {
 5.          WarehouseRecords[j].Add(ItemRecords[i]);
 6.          WarehouseRecords[j].Add(ItemRecords[n – 1 – i]);
 7.          j = (j + 1) % m;
 8.    }
} //End  of UDD
```

Приведенный алгоритм относится к классу жадных алгоритмов и распределяет объекты по хранилищам так, как это показано на рисунке 3.6. На каждом шаге выполнения алгоритма в очередное хранбилище помешаются два объекта, симметричные относительно середины массива (их индексы задаются в строках 4, 5). Индекс хранилища определяется как остаток от деления текущего индекса массива ItemRecords на количество хранилищ. Таким образом, в хранилище помещаются наибольший и наименьший из нераспределенных объектов. Это отличает предложенный алгоритм от большинства подобных алгоритмов.

Замечание. Если n *нечетное число, то последняя пара объектов помещается в очередное хранилище после окончания цикла.*

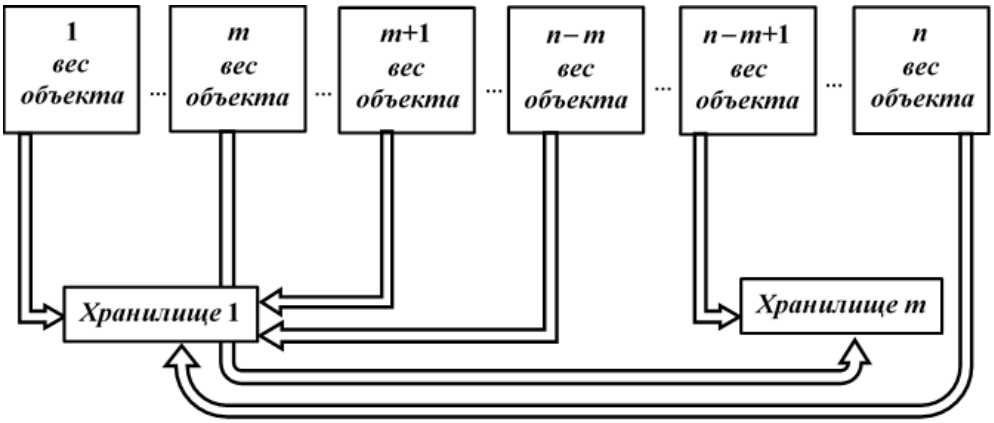


Рис. 3.6. Распределение объектов по хранилищам

Вычислительная сложность алгоритма складывается из вычислительной сложности операций сортировки массива объектов и однократного просмотра этого массива, следовательно, имеет порядок $O(n^2+n)$.

Поскольку предложенный алгоритм относится к классу эвристических жадных алгоритмов, то оценить его качественные характеристики можно только экспериментальным путем. Поэтому для анализа алгоритма оптимального распределения был проведен ряд экспериментов. Цель этих экспериментов состояла в том, чтобы определить зависимости:

– времени выполнения алгоритма от количества объектов;

– качества распределения (разности между максимумом и минимумом заполнения хранилищ) от количества хранилищ и интервала значений весов объектов.

Эксперименты проводились на рабочей станции с характеристиками: процессор Intel® Core™ i7-8700K и 32 ГБ оперативной памяти.

В этой серии экспериментов количество хранилищ изменялось по степеням 2, от 2^3 до 2^{16}. Количество объектов изменялось от $12\cdot10^6$ до $24\cdot10^7$ с шагом $12\cdot10^6$. Веса объектов изменялись в интервалах от w+1 до w+20000 (w=0, 20000, 40000, …, 100000) и в интервалах от w+1 до $w+2\cdot10^6$ (w=0, $2\cdot10^6$, $4\cdot10^6$, …, $10\cdot10^6$). Для количества объектов приведены результаты только для граничных значений, а для весов еще и для одного промежуточного значения.

В каждом эксперименте производилось распределение объектов, веса которых находились в заданном интервале между заданным числом хранилищ. Затем определялись величины W_{max} и W_{min} для наиболее и наименее заполненных хранилищ. Затем по формуле $\frac{W_{max}-W_{min}}{\frac{W_{max}+W_{min}}{2}}\times 100$ вычислялась величина (в процентах), характеризующая качество распределения.

Оптимизация осуществлялось на основе целевой функции (1) без ограничений

Далее приводятся результаты экспериментов.

Эксперимент1. В этом эксперименте проводился анализ поведения алгоритма при следующих условиях: количество объектов равно $12\cdot10^6$, наибольшее значение веса объекта отличается от наименьшего не более, чем на 20000 (минимум) и $2\cdot10^6$ (максимум) единиц. Результат эксперимента приведен в таблице 3.7.

Таблица 3.7. Качество распределения для $12\cdot10^6$ объектов

n	**12000000**					
m	**Интервалы значений объемов классов эквивалентности**					
	1-20000	**60001-80000**	**100001-120000**	**10^6-$2\cdot10^6$**	**$6\cdot10^6$-$8\cdot10^6$**	**$10\cdot10^6$-$12\cdot10^6$**
8	0,00000071	0,00000010	0,00000006	0,00000007	0,00000001	0,00000001
16	0,00000181	0,00000017	0,00000018	0,00000010	0,00000002	0,00000002
32	0,00000605	0,00000069	0,00000041	0,00000026	0,00000004	0,00000003
64	0,00001056	0,00000146	0,00000073	0,00000035	0,00000008	0,00000005
128	0,00001632	0,00000274	0,00000111	0,00000116	0,00000016	0,00000006
256	0,00429285	0,00427032	0,00426881	0,00426939	0,00426693	0,00426678
512	0,00858716	0,00853996	0,00853647	0,00853728	0,00853391	0,00853374
1024	0,01715141	0,01707737	0,01707183	0,01707426	0,01706710	0,01706700
2048	0,03426990	0,03415179	0,03414767	0,03414707	0,03413712	0,03413654
4096	0,06847980	0,06830988	0,06829985	0,06830409	0,06828775	0,06828404
8192	0,13685820	0,13656729	0,13654501	0,13656129	0,13652577	0,13652495

16384	0,27337972	0,27292487	0,27291490	0,27292661	0,27286646	0,27285650
32768	0,54579613	0,54509837	0,54504163	0,54515903	0,54498541	0,54497094
65536	1,09419349	1,09310503	1,09302175	1,09326761	1,09295563	1,09292142

Приведенные в таблице результаты эксперимента показывают, что алгоритм распределения дает достаточно хорошие результаты. При таком распределении время выполнения операции Join будет примерно одинаковым на всех процессорах. Время работы алгоритма составило в среднем 1,47 секунды.

Эксперимент 2. В этом эксперименте проводился анализ поведения алгоритма при следующих условиях: количество объектов равно $24 \cdot 10^7$, наибольшее значение веса объекта, как и в эксперименте 1, отличается от наименьшего не более, чем на 20000 (минимум) и $2 \cdot 10^6$ (максимум) единиц. Результат эксперимента приведен в таблице 3.8.

Таблица 3.8. Качество распределения для $24 \cdot 10^7$ объектов

n	**240000000**					
m	**Интервалы значений объемов классов эквивалентности**					
	1-20000	**60001-80000**	**100001-120000**	**10^6-$2 \cdot 10^6$**	**$6 \cdot 10^6$-$8 \cdot 10^6$**	**$10 \cdot 10^6$-$12 \cdot 10^6$**
8	0,00000005	0,00000001	0,00000001	0,00000001	0,00000001	0,00000000
16	0,00000011	0,00000002	0,00000001	0,00000001	0,00000001	0,00000000
32	0,00000024	0,00000003	0,00000002	0,00000001	0,00000001	0,00000000
64	0,00000039	0,00000005	0,00000003	0,00000005	0,00000001	0,00000000
128	0,00000072	0,00000010	0,00000010	0,00000008	0,00000001	0,00000001
256	0,00000172	0,00000016	0,00000024	0,00000014	0,00000002	0,00000001
512	0,00000378	0,00000034	0,00000037	0,00000019	0,00000003	0,00000001
1024	0,00085758	0,00085390	0,00085385	0,00085362	0,00085338	0,00085336
2048	0,00171473	0,00170758	0,00170751	0,00170721	0,00170674	0,00170672
4096	0,00342762	0,00341466	0,00341445	0,00341414	0,00341347	0,00341346
8192	0,00685273	0,00682862	0,00682956	0,00682784	0,00682679	0,00682677
16384	0,01369133	0,01365629	0,01365698	0,01365509	0,01365303	0,01365300
32768	0,02736578	0,02730840	0,02730973	0,02730736	0,02730416	0,02730405

65536	0,05468587	0,05460827	0,05460853	0,05460651	0,05460057	0,05460054

Результаты этого эксперимента подтверждают выводы, сделанные на основе результатов предыдущего эксперимента. Более того, прослеживается тенденция улучшения качества распределения с увеличением количества объектов. Время работы алгоритма в этом эксперименте составило в среднем 33,1 секунды.

Эксперимент 3. В этом эксперименте была исследована возможность параллельной реализации алгоритма распределения. Был использован параллельный алгоритм быстрой сортировки. Распараллеливание было проведено на двенадцати потоках (двенадцать виртуальных ядер процессора). Результаты распараллеливания приведены в таблице 3.9 и на рисунке 2.

Таблица 3.9. Параллельное и последовательное выполнение алгоритма распределения

n	$T_{последовательно}$	$T_{параллельно}$
$12 \cdot 10^6$	1,47	0,82
$60 \cdot 10^6$	8,70	3,63
$108 \cdot 10^6$	15,94	8,69
$156 \cdot 10^6$	23,17	11,37
$204 \cdot 10^6$	30,40	14,22
$252 \cdot 10^6$	37,63	19,52
$300 \cdot 10^6$	44,87	22,18
$348 \cdot 10^6$	50,19	26,40

Эксперимент показал, что распараллеливание несущественно, всего лишь в два раза, улучшает временные характеристики алгоритма. Учитывая малое время последовательного выполнения алгоритма, можно сделать вывод о том, что в этом случае распараллеливание нецелесообразно.

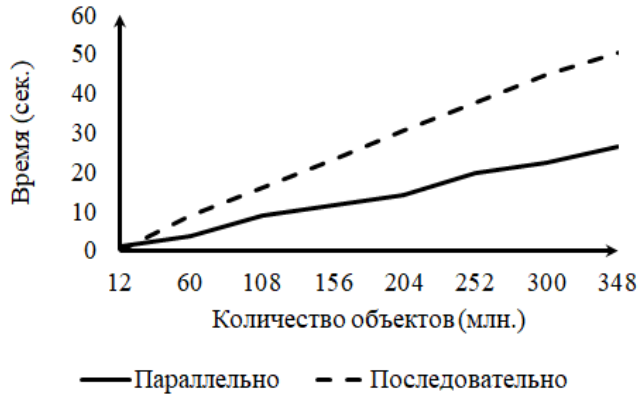


Рис. 3.7. Две реализации алгоритма распределения

Результаты обоих экспериментов показали, что предложенный алгоритм имеет хорошую скорость вычислений и дает достаточно качественное распределение объектов по хранилищам. Рассмотренные объемы данных, распределение которых осуществлялось, несомненно относятся к классу big data. Поэтому возможность выполнить распределение данных за малое время и с достаточно хорошей балансировкой объёмов данных (различие составляет не более 2%) способно обеспечить эффективное использование параллельных вычислительных систем (машин баз данных).

Алгоритм, реализующий симметричное горизонтальное распределение файлов операндов X_K и Y_K по N процессам P_1, …, P_N, состоит из следующих шагов:

1. Из файлов $indX_K$ и $indY_K$ формируется файл параметров распределения $indD = indX_K \cap indY_K = (< K^*_{(1)}, I_{1X}, m_{1X}, m_{1Y}, I_{1Y}, R_1 >, ..., < K^*_{(p)}, I_{pX}, m_{pX}, I_{pY}, m_{pY}, R_p >)$, где $R_j = m_{jX} \times m_{jY}$, $j=(1, \ldots, p)$.

2. Файл $indD$ упорядочивается по возрастанию (или убыванию) произведений R_j.

3. Формируются два курсора для упорядоченного файла $indD$: $C_a = 1$ и $C_d = p$.

4. Выполняется $\frac{p}{N}$ итераций, каждая из которых состоит из следующих шагов:

4.1. N пар классов эквивалентности файлов-операндов, определяемых записями файла *indD* с номерами C_a, …, $C_a + N - 1$, последовательно распределяются между N процессами, начиная с первого процесса.

4.2. N пар классов эквивалентности файлов-операндов, определяемых записями файла *indD* с номерами C_d, …, $C_d - N + 1$, последовательно распределяются между N процессами, начиная с первого процесса.

4.3. Вычисляются новые значения курсоров по формулам: $C_a = C_a + N$ и $C_d = C_d - N$.

5. Пункты 4.1-4.4 повторяются до тех пор, пока весь файл *indD* не будет просмотрен, то есть не будет достигнуто равенство $C_a = C_d$.

6. Если число N нечетно, то классы эквивалентности файлов X_K и Y_K, соответствующие записи файла *indD* с номером $\left\lceil \frac{N}{2} \right\rceil + 1$, добавляются к фрагменту, которому соответствует наименьшее количество формируемых записей файла-результата Z_K.

В таблице 3.10 приведен пример формирования файла *indD* из индексных файлов $indX_K$ и $indY_K$. Эти файлы содержат четыре записи с одинаковыми значениями множества ключей K. В двух последних столбцах таблицы приведены фрагменты записей файла *indD*. Эти фрагменты содержат значение экземпляра множества ключей и количество формируемых записей файла-результата.

Таблица 3.10. Формирование индексного файла для симметричного горизонтального распределения файлов-операндов

$indX_K$			$indY_K$			$indX_K \cap indY_K$	
$K^*_{(j)}$	I_{jX}	m_{jX}	$K^*_{(j)}$	I_{jY}	m_{jY}	$K^*_{(j)}$	R_j
$K^*_{(0)}$	0	90	$K^*_{(0)}$	0	65	$K^*_{(0)}$	5850
$K^*_{(1)}$	90	85	$K^*_{(2)}$	65	110	$K^*_{(4)}$	27000
$K^*_{(4)}$	175	120	$K^*_{(4)}$	175	225	$K^*_{(7)}$	9834

$indX_K$			$indY_K$			$indX_K \cap indY_K$	
$K^*_{(5)}$	295	63	$K^*_{(6)}$	400	80	$K^*_{(8)}$	20375
$K^*_{(7)}$	358	149	$K^*_{(7)}$	480	66		
$K^*_{(8)}$	507	163	$K^*_{(8)}$	446	125		
$K^*_{(9)}$	670	59	$K^*_{(11)}$	571	129		

В таблице 3.11 приведены три примера, показывающие работу алгоритма распределения для восемнадцати фрагментов файлов операндов, распределяемых между тремя процессами для различных интервалов изменения величины R_j. В этих примерах показано, что величина $R_{\max} - R_{\min}$ составляет незначительный процент от среднего числа формируемых процессами записей выходного файла. Следовательно, время выполнения каждого процесса не будет значительно отличаться от времен выполнения остальных процессов.

Таблица 3.11. Результаты распределения классов эквивалентности файлов- операндов по трем процессам

Пример 1 $1 \le R_j \le 100$				Пример 2 $1 \le R_j \le 1000$				Пример 3 $1 \le R_j \le 10000$			
R_j	P_1	P_2	P_3	R_j	P_1	P_2	P_3	R_j	P_1	P_2	P_3
98	290	276	285	7145	2569	2444	2594	7145	20670	19167	20366
94	Среднее:			813	Среднее:			6121	Среднее:		
93	284			736	2536			5577	20068		
74	$R_{\max} - R_{\min}$:			650	$R_{\max} - R_{\min}$:			5526	$R_{\max} - R_{\min}$:		
67	14			627	150			4576	150		
63	5%			609	6%			4483	8%		
61				512				4348			
49				482				4201			
43				444				4081			

43		422		3871	
37		291		2593	
35		278		2581	
31		243		1473	
18		193		1135	
16		178		1013	
12		140		881	
11		38		541	
6		13		57	

В таблице 3.12 приведены два примера, показывающие работу алгоритма для ста фрагментов файлов операндов, распределяемых между пятью процессами для различных интервалов изменения величины R_j.

Как и в предыдущих примерах, различие между размерами формируемых разными процессами фрагментов файла-результата незначительно.

Таблица 3.12. Результаты распределения классов эквивалентности файлов- операндов по пяти процессам

P_j	R_j	Max R	Min R	Среднее значение R_j	$R_{\max} - R_{\min}$	%
$1 \le R_j \le 100$						
	1437309	1596437	1437309	1512916	159128	11
	1459158					
	1510585					
	1561092					
	1596437					
$1 \le R_j \le 1000$						
P1	21155066	23536514	21155066	22448839	2381448	11
P2	22072361					
P3	22760445					

P4	22719811	
P5	23536514	

Таким образом, можно утверждать, что эвристический алгоритм обеспечивает достаточно равномерную загрузку процессов.

На втором этапе каждый процесс выполняет операцию слияния нестрого упорядоченных файлов над связанными с ним фрагментами файлов-операндов. Завершается второй этап слиянием всех полученных фрагментов файла-результата в один файл.

3.6. Алгоритм параллельной реализации операции соединения в реляционной модели SQL

Как было показано в главе 2 существует, по крайней мере, гомоморфное соответствие между файловой моделью данных и реляционной моделью SQL. Тогда строго упорядоченным файлам X_L и Y_M соответствуют таблицы P и Q во второй или третьей нормальной форме по составным ключам, соответствующим множествам ключей L и M. Операции слияния нестрого упорядоченных файлов X_L и Y_M по множеству ключей K соответствует операция P INNER JOIN Q ON $\pi(P.K, Q.K)$. На практике операция JOIN реализуется при следующих условиях, которые соответствуют требованиям к операции слияния нестрого упорядоченных файлов:

– Таблицы P и Q имеют разные схемы, в каждую из которых входит простой или составной ключ K.

– Предикат π определен на множествах экземпляров ключа K в таблицах P и Q и имеет вид P.K = Q.K.

В общем случае, файлы X_L и Y_M нестрого упорядочены, а соответствующие им таблицы $P.K$ и $Q.K$ будут в первой нормальной форме. В этом случае, таблицы, как и файлы, состоят из классов эквивалентности, в которых все строки содержат один и тот же экземпляр ключа K. Тогда для параллельной реализации операции JOIN можно использовать метаданные, определяющие распределение строк таблицы по классам эквивалентности, основанное, как и ы случае

файловой модели данных, на индексно-последовательном методе доступа (ISAM).

Пусть K_1, …, K_n – множество значений ключа *K*. Таблицы *P*(*K*, …) и *Q*(*K*, …) имеют схемы, которые содержат ключ *K* и произвольные наборы полей (как правило, различные). Таблице *P* можно поставить в соответствие индексную таблицу со схемой *indP*(*K*, *I*, *M*). Эта таблица – есть множество строк $indP = \{< K_1^*, i_1, m_1 >, ..., < K_m^*, i_m, m_m >.$ i_j – индекс первой записи класса эквивалентности, строки которого содержат экземпляр K_j^* ключа *K*, m_j – количество записей в этом классе эквивалентности. Поле *I* имеет тип Счетчик строк, а поле *M* – целочисленное. Таблица *indP* может быть получена в результате двух запросов:

1. *Q*1=INSERT INTO *indPt* (*K*, *M*) SELECT *P.K*, 1 AS *M* FROM *P* ORDER BY *P.K*;
2. *Q*2=SELECT *indPt.K*, First(*indPt.M*) AS *I*, Sum(*IndPt.M*) AS *M* INTO *indP* FROM *indPt* GROUP BY *indPt.K* ORDER BY *indPt.K*;

Запрос *Q*1 формирует промежуточную таблицу, которая содержит столько же строк, что и таблица *P*. Каждая строка этой таблицы содержит свой номер, значение ключа *K* из соответствующей строки таблицы *P* и поле *M*, содержащее значение 1. Запрос *Q*2 завершает построение таблицы *indP* посредством операции группировки. Этому двухпроходному алгоритму соответствует простой однопроходный алгоритм, который легко разработать на любом языке программирования, связанным с СУБД, или на языках манипулирования данными, такими как Tranzact-SQL или PL-SQL. Листинг 1 демонстрирует реализацию однопроходного алгоритма построения индексного файла для индексно-последовательного доступа к таблице *P*.

Листинг 3.2. Построение индексного файла средствами языка Tranzact-SQL

```
1.  DECLARE @K <тип>, @currentK <тип>, @I int, @M int, @RowCount int
2.  DECLARE @j int
3.  DELETE FROM indP
4.  DECLARE cursorP CURSOR FOR SELECT K FROM P ORDER BY K
5.  OPEN cursorP
6.  SET @RowCount =@@Cursor_Rows
```

```
7.   SET @j=0
8.   FETCH NEXT FROM cursorP INTO @K
9.   SET @currentK =@K
10.  SET @I=1
11.  SET @M=1
12.  WHILE @j<@ RowCount –1
13.  BEGIN
14.      FETCH NEXT FROM cursorP INTO @K
15.      IF @currentK =@K
16.          SET @M=@M+1
17.      ELSE
18.      BEGIN
19.          INSERT INTO indP (K, I, M) VALUES (@currentK, @I, @M)
20.          SET @currentK =@K
21.          SET @I=@I+@M
22.          SET @M=1
23.      END
24.      SET @j=@j+1
25.  END
26.  INSERT INTO indP (K, I, M) VALUES (@currentK, @I, @M)
```

Подготовительная часть алгоритма (строки 8-11) формирует заготовку первой строки таблицы *indP*. Эта строка содержит первое по порядку значение ключа *K*. Основная часть алгоритма заключена в теле цикла (строки 13-25). Здесь, при совпадении значения ключа *K* в очередной строке таблицы *P* и в заготовке текущей строки таблицы *indP*, накапливается значение поля *M*, в противном случае производится вывод текущей строки в таблицу *indP* и формируется заготовка новой строки этой таблицы. По завершении цикла, последняя сформированная строка выводится в таблицу *indP* (строка 26).

Таблица *indQ* для таблицы *Q* может быть получена аналогично.

Очевидно, что таблица-результат операции JOIN будет содержать только те классы эквивалентности, которые принадлежат пересечению индексных таблиц *indP*∩ *indQ*. Запрос, в результате которого получается это пересечение, имеет вид:

```
SELECT indP.M*indQ.M AS MM, indP.K, indP. I, indQ.I,  indP.M, indQ.M
    INTO ComInd FROM indP, indQ
    WHERE indP.K = indB.K
    ORDER BY indP.M*indQ.M
```

Результат запроса – таблица *ComInd*. Она содержит составной ключ *K*, поля *I* и *M* обеих индексных таблиц, а также поле *MM*, по которому она упорядочена. Поле *MM* вычисляется как произведение количества строк в обоих классах эквивалентности, строки которых содержат одно и то же значение составного ключа *K*. Его значение определяет число выходных строк, которое получится в результате обработки этих классов эквивалентности операцией JOIN.

Использование таблицы *ComInd* позволяет распределить таблицы *P* и *Q* между несколькими независимыми базами данных. Если имеется *N* таких баз данных, то эти таблицы разделяются на строки в таблицу *indP* фрагментов. Каждая база данных $DBSnp_i$ (i=1, …, N) содержит пару фрагментов $<P_i, Q_i>$ таблиц *P* и *Q*. При этом классы эквивалентности, записи которых содержат одинаковые значения составного ключа *K*, полностью расположены в одном и только одном фрагменте. Такой способ распределения таблиц соответствует принципу симметричного горизонтального распределения данных. Таблица *ComInd* соответствует файлу *indD*. Алгоритм, реализующий симметричное горизонтальное распределение таблиц *P* и *Q* также может быть реализован средствами языка Tranzact-SQL так, как это показано в листинге 3.2.

Листинг 3.3. Реализация симметричного горизонтального распределения средствами языка Tranzact-SQL

```
1.  DECLARE @KA <тип>, @KD <тип>, @RowCount int, @i int
2.  DECLARE cursorASC CURSOR FOR SELECT K, MM FROM ComInd
                                        ORDER BY MM ASC
3.  DECLARE cursorDSC CURSOR FOR SELECT K, MM FROM ComInd
                                        ORDER BY MM DESC
4.  OPEN cursorASC
5.  OPEN cursorDSC
6.  SET @ RowCount =@@Cursor_Rows
7.  IF @ RowCount %2=0
8.  SET @RowCount =@RowCount /2
9.  ELSE
10. SET @RowCount =@RowCount /2+1
11. SET @i=0
12. WHILE @i<@RowCount
13. BEGIN
14.     FETCH NEXT FROM cursorASC INTO @ KA
15.     FETCH NEXT FROM cursorDSC INTO @ KD
16.     IF @i % N=0
```

```
17.       BEGIN
18.          INSERT INTO PSnp1 SELECT * FROM P WHERE K=@KA
19.          INSERT INTO PSnp1 SELECT * FROM P WHERE K=@KD
20.          INSERT INTO RSnp1 SELECT * FROM R WHERE K=@KA
21.          INSERT INTO RSnp1 SELECT * FROM R WHERE K=@KD
22.       END
23.        …
24.       IF @i % N=N–1
25.       BEGIN
26.           INSERT INTO PSnpN SELECT * FROM P WHERE K=@KA
27.           INSERT INTO PSnpN SELECT * FROM P WHERE K=@KD
28.           INSERT INTO RSnpN SELECT * FROM R WHERE K=@KA
29.           INSERT INTO RSnpN SELECT * FROM R WHERE K=@KD
30.       END
31.       SET @i=@i+1
32.   END
```

Средства языка Tranzact-SQL позволяют существенно использовать в процедуре, реализующей алгоритм, упорядоченность таблиц, что отличает его от языка SQL, отражающего свойства классической реляционной модели. Поэтому строки таблицы *ComInd* могут быть прочитаны двумя запросами по возрастанию и по убыванию значений поля *MM* (строки 2, 3). Это позволяет соединить наибольшее и наименьшее значения этого поля. В зависимости от того четно или нечетно число строк в таблице *ComInd*, определяется количество обрабатываемых строк (строки 7-10), поскольку считывание таблицы двумя запросами с противоположным упорядочиванием позволяет читать только половину таблицы (плюс одна строка, если число строк нечетно). В теле основного цикла (строки 14-31) формируются фрагменты таблиц *P* и *Q*. Как и в случае файловой модели, после выполнения этого варианта алгоритма, количества строк таблицы-результата, получаемые в результате операции JOIN над фрагментами P_i и R_i таблиц *P* и *R*, будут минимально отличаться друг от друга.

Разработка процедур, реализующих симметричное горизонтальное распределение таблиц, может быть легко автоматизирована. Прикладной программист, используя метаданные, может указать на связи между ключами таблиц-операндов операции JOIN и на основе этих связей система автоматизации программирования построит все необходимые хранимые процедуры на языке манипулирования данными (Transact SQL, PG/SQL, PL/SQL), разместит их в соответствующем разделе БД и выполнит компиляцию [187].

После завершения симметричного горизонтального распределения таблиц P и Q по базам данных $DBSnp_i$ над содержащимися в этих базах данных парами фрагментов таблиц P_i и Q_i выполняются операции JOIN, по завершении которых полученные фрагменты таблицы результата сливаются в одну таблицу.

3.7. Стратегия повышения эффективности процессов МОД

На основании рассмотренных методов возможно построение общей стратегии повышения эффективности процессов МОД. Цель этой стратегии – разработка эффективных запросов к базе данных при следующих условиях:

– структура данных определена заранее и нечасто подвергается изменениям;

– объемы данных в базе очень велики (относительно используемой вычислительной техники);

– запросы после разработки используются достаточно длительное время;

– запросы имеют большую вычислительную сложность, а время их выполнения критично.

В этом случае, затраты на построение запроса к базе данных, представленной в одной из рассматриваемых моделей, будут окупаться за счет длительного периода эксплуатации этого запроса.

Предлагаемая стратегия основана на том, что метаданные, которые описывают свойства хранящихся в базе данных, также незначительно подвергаются изменениям и могут быть использованы как основа для повышения эффективности запросов.

Стратегия состоит из следующих этапов.

1. Выбирается модель данных уровня проектирования. Такой моделью может быть реляционная модель, модель "сущность–связь" или любая другая модель, пригодная для проектирования баз данных.

2. В выбранной модели разрабатывается архитектура базы данных: агрегаты данных, связи между ними, запросы.

3. Проводится анализ данных и выбирается одна из промежуточных моделей по следующему правилу: если реальные исходные данные таковы, что агрегаты

данных содержат практически все значения ключей, то выбирается многомерно-матричная модель, в противном случае, выбирается файловая модель данных. В первом случае агрегатам данных соответствуют многомерные матрицы, содержащие незначительное количество нейтральных элементов (не разреженные), во втором – разреженные.

4. Выражения запросов транслируются из представления на языке модели данных, использованной для проектирования, в язык выбранной промежуточной модели данных.

5. Производится оптимизация оттранслированных выражений запросов либо методом синтеза эффективных выражений запросов (построение алгебраических выражений, содержащих те же операнды и приводящих к тому же результату), либо преобразования имеющихся выражений запросов в выражения, порождающие более эффективные процессы обработки данных.

6. Выбирается модель вычислений и алгоритмы составляющих запросы операций, которые соответствуют выбранной модели вычислений и эффективно в ней реализованы.

Реализация предложенной стратегии повышения эффективности процессов МОД может осуществляться как "вручную", так, и автоматизировано, с использованием средств трансляции выражений запросов и программ, реализующих оптимизацию этих выражений по заданным критериям.

3.8. Заключительные замечания к главе 3

В главе рассмотрены известные методы оптимизации запросов на обработку данных.

Проведено доказательство на основе применения принципа инвариантного погружения того, что для синтеза оптимизированного алгебраического выражения запроса на МОД может быть использован метод динамического программирования. На основе экспериментального анализа показано, что особенности процессов МОД позволяют эффективно использовать этот метод оптимизации.

Проведено обобщение алгоритма Кэннона параллельного умножения матриц на случай (λ, μ)-свернутого произведения многомерных матриц.

Сделан анализ алгоритмов, реализующих операцию слияния нестрого упорядоченных файлов.

Предложен способ параллельной реализации этой операции на основе принципа симметричного горизонтального распределения файлов-операндов.

Приведено описание алгоритма, реализующего этот тип распределения данных. Показано как этот алгоритм может быть реализован в реляционных СУБД.

Предложена стратегия повышения эффективности процессов МОД.

Основные результаты, полученные в данной главе, были опубликованы в работах [74, 117, 158, 162-164, 174-177].

Глава 4. АРХИТЕКТУРЫ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ ДЛЯ МАССОВОЙ ОБРАБОТКИ ДАННЫХ

4.1. Этапы построения программно-аппаратных комплексов

В постановлении Правительства Российской Федерации от 28 декабря 2022 г. № 2461 [195] дано следующие определение программно-аппаратного комплекса:

«Программно-аппаратный комплекс» – комплекс технических и программных средств (программного обеспечения), работающих совместно для выполнения одной или нескольких специальных задач, являющийся электронной вычислительной машиной или специализированным электронным устройством (устройствами), функционально-технические характеристики которого (которых) определяются исключительно совокупностью программного обеспечения и технических средств и не могут быть реализованы при их разделении.

Программно-аппаратный комплекс является самостоятельно используемым, законченным техническим изделием, имеющим серийный номер.»

Программно-аппаратный комплекс – это уточнение более общего понятия "вычислительный комплекс", который определяется как взаимосвязанная совокупность средств вычислительной техники, в которую входит не менее двух процессоров, объединенных системой управления, имеющих общую память, единое программное обеспечение и общие периферийные устройства. Уточнение состоит в том, что программно-аппаратный комплекс определяется как техническое решение концепции алгоритма работы сложной системы, управление которой осуществляется, как правило, исполнением кода из определённого базового набора команд (системы команд). То есть, это набор технических и программных средств, работающих совместно для выполнения одной или нескольких сходных задач.

Некоторые разработчики [196] считают, что наиболее перспективный подход при разработке компактных высокопроизводительных вычислительных комплексов основывается на концепции построения реконфигурируемых мно-

гопроцессорных вычислительных систем. Суть этой концепции заключается в том, что архитектура вычислительной системы должна иметь возможность адаптироваться под структуру решаемой задачи. Фактически это означает, что пользователю должна быть предоставлена возможность программировать проблемно-ориентированные многопроцессорные вычислительные системы, структура которых адекватна решаемой ими задаче. При этом достигается высокая реальная производительность вычислительной системы на широком классе задач, а также почти линейный рост производительности при увеличении числа процессоров. В отличие от многопроцессорных вычислительных систем с «жесткой» архитектурой, в частности, кластерных суперЭВМ, архитектура реконфигурируемых систем может изменяться в процессе ее функционирования.

С понятием программно-аппаратного комплекса тесно связано понятие модели вычислений, которая определяется как сочетание множества допустимых операций, используемых для вычисления и относительных издержек, связанных с их применением. Далее будут использованы модели вычислений для параллельного программирования, основанные на:

– многопоточности (multithreading), когда процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся параллельно;

– обмене сообщениями (message passing), когда порожденные и выполняющиеся процессы взаимодействуют, посылая и получая сообщения;

– параллелизме данных (data parallelism), который заключается в применении одной и той же операции к множеству элементов структур данных, то есть реализует архитектуру SPMD (одна программа много данных) [197];

– общей памяти (shared memory), когда все процессы совместно используют общее адресное пространство, к которому они асинхронно обращаются с запросами на чтение и запись;

– потоках данных (dataflaw), когда используется ассоциативное распределение ресурсов вычислительного комплекса.

Таким образом, построение программно-аппаратного комплекса, ориентированного на решение задач МОД, состоит из следующих этапов:

1. Выбор одной из трех (файловой, реляционной или многомерно-матричной) модели данных на основе свойств данных, которые присущи решаемой задаче. Основное свойство данных, определяющее выбор модели, – это свойство "плотности" ключей. Если во всех реальных случаях, когда решается задача, данные идентифицируются практически всеми значениями ключей, целесообразно использовать многомерно-матричную модель данных. Если же используется незначительная часть значений большинства ключей, что приводит к высокой степени разреженности многомерных матриц, более эффективным будет использование файловой модели данных.

2. Выбор на основе выбранной модели данных аппаратной архитектуры вычислительного комплекса. Далее будет показано, что для многомерно-матричной модели в большей степени подходят комплексы, основанные на SMP-архитектуре, а для файловой модели – комплексы, сочетающие свойства SMP- и MPP-архитектур.

3. На основе выбранной модели данных определяется модель вычислений, то есть определяется набор операций МОД, выбираются (или при необходимости разрабатываются) алгоритмы реализации операций, оценивается вычислительная сложность этих алгоритмов.

4. На основе выбранных модели данных и архитектуры программно-аппаратного комплекса "собирается" из имеющихся вычислительных средств действующая модель программно-аппаратного комплекса. Определяются операционные системы, системы программирования и системы управления базами данных.

5. Разрабатывается программное обеспечение, реализующее алгоритмы распределения данных и взаимодействия процессоров и потоков команд и данных. Разрабатывается прикладное программное обеспечение, реализующее операции и процессы МОД для конкретного класса задач.

Эти этапы реализуются на основе предложенных далее архитектур и технологий построения программно-аппаратных комплексов для МОД [117].

4.2. Архитектура программно-аппаратного комплекса для реализации многомерно-матричной модели данных

При построении архитектуры программно-аппаратного комплекса для реализации многомерно-матричной модели данных следует учитывать тот факт, что основная сложность состоит в реализации операции умножения многомерных матриц. Поэтому далее рассматривается архитектура программно-аппаратного комплекса, ориентированного на реализацию именно этой операции.

Из утверждения 3.1 (п. 3.4.1) следует, что при наличии скоттовых индексов $s_1, ..., s_\lambda$ программно-аппаратный комплекс должен состоять из $\prod_{i=1}^{\lambda} d(s_i)$ независимых блоков, где $d(s_i)$ – произведение размерностей скоттовых индексов матриц-операндов. Каждый такой блок имеет индекс $s_1^* ... s_\lambda^*$, который состоит из фиксированных значений скоттовых индексов, и вычисляет значение сечения матрицы результата как произведение сечений матриц-операндов. Все эти сечения простые и имеют ориентацию $(s_1, ..., s_\lambda)$. Поэтому архитектура блока – это $(p+q-2\lambda-2\mu)$-решетка, в памяти которой расположены соответствующие $(p-\lambda)$-кратное сечение матрицы *A*, $(q-\lambda)$-кратное сечение матрицы *B* и $(r-\lambda)$-кратное сечение матрицы *C*. Для вычисления произведения сечений матриц *A* и *B* используется программное обеспечение, основанное на одном из алгоритмов параллельного умножения многомерных матриц, например, алгоритм, предложенный в п. 3.4.2.

Пример 4.1. В этом примере [175] рассматривается программно-аппаратный комплекс для реализации операции умножения трехмерных матриц для условий, рассмотренных в примерах 3.4 и 3.5, то есть, E=3, $A = \|A_{lsc}\|$, $B = \|B_{scm}\|$ и $C = \|C_{lsm}\|$ – трехмерные матрицы, индексы которых при-

нимают значения от 0 до 2, а элементы – блоки исходных трехмерных матриц и матрицы-результата [175].

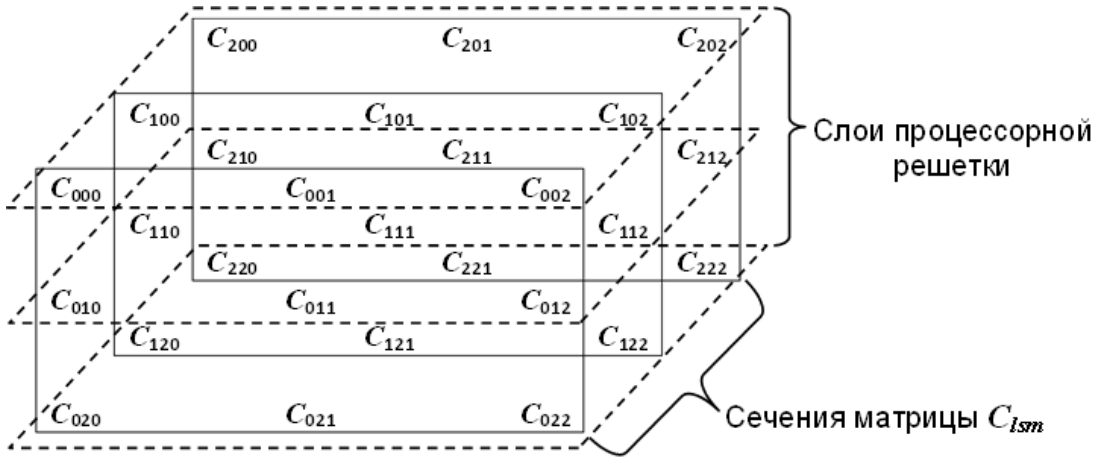


Рис. 4.1. Распределение блоков матрицы C_{lsm} по слоям процессорной решетки

Таким образом, для умножения трехмерных матриц необходима аппаратная архитектура – трехмерная решетка, которая логически разделена на двумерные решетки, в каждой из которых выполняется параллельное умножение плоских матриц – двумерных сечений трехмерных матриц-операндов. Количество двумерных сечений не превосходит величину n_s – количество значений скоттова индекса s. Программное обеспечение, каждого процессора в решетке включает программную реализацию алгоритма умножения блоков сечений трехмерных матриц, подобного тому, который приведен в пунктах 3.4.1, 3.4.2. Очевидно, что для эффективной реализации передачи блоков сечений матриц-операндов целесообразно использовать не двумерную решетку, а тор. Таким образом, программно-аппаратный комплекс для реализации (1, 1)-свернутого произведения трехмерных матриц представляет собой вектор тороидальных слоев трехмерной решетки (рисунок 4.2).

В случае, когда в матрицах-операндах два или более скоттовых индексов архитектура программно-аппаратного комплекса представляет решетку, размерность которой определяется количеством скоттовых индексов. В узлах этой решетки располагаются процессоры, реализующие параллельное умножение сечений матриц-операндов.

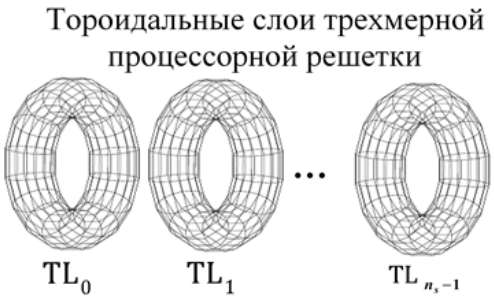


Рис. 4.2. Программно-аппаратный комплекс для умножения трехмерных матриц

На рисунке 4.3 показана архитектура программно-аппаратного комплекса для реализации (2,1)-свернутого произведения четырехмерных матриц, у которых по одному свободному индексу, один кэлиев индекс и две скоттовых индекса s_1 и s_2.

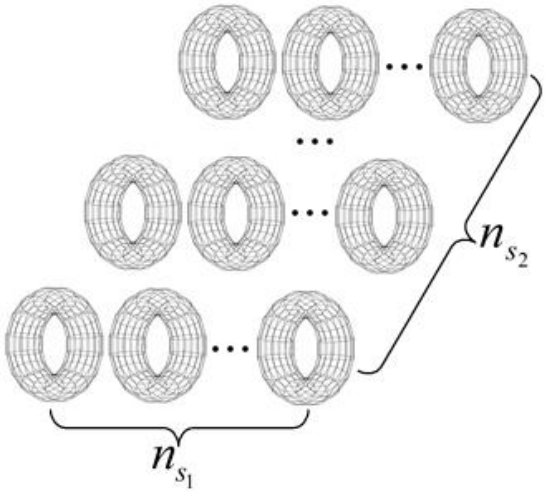


Рис. 4.3. Программно-аппаратный комплекс для умножения четырехмерных матриц с двумя скоттовыми индексами

В рассмотренном примере матрицы-операнды имели только один свободный индекс, сечения которых по скоттовым индексам – плоские матрицы. Поэтому для их параллельного умножения использовалась двумерная решетка или тор. Если матрицы-операнды имеют больше одного свободного индекса $l = (l_1,...,l_\kappa), (\kappa > 1)$ и $m = (m_1,...,m_\nu), (\nu > 1)$, то в этом случае в результате умножения сечений этих матриц по скоттовым индексам получаются $(\kappa+\nu)$-мерные сечения матрицы результата. Поэтому архитектура процессора, реализующего параллельное умножение сечения таких матриц-операндов представляет собой $(\kappa+\nu)$-мерную решетку или соответствующий ей тор. Программное

обеспечение такого процессора реализует умножение блоков сечений матриц-операндов и сложение этих произведений на основе алгоритма из пункта 3.4.2.

4.3. Архитектуры программно-аппаратных комплексов для реализации простых операций теоретико-множественной модели данных

В этом разделе рассматриваются архитектуры программно-аппаратных комплексов, ориентированных на параллельную реализацию простых операций (сортировка, выборка, свертка и слияние строго упорядоченных файлов), определенных в теоретико-множественной (файловой) модели. Все эти операции допускают распараллеливание, но архитектуры программно-аппаратных комплексов, реализующих эти операции, как правило, различны.

4.3.1. Параллельная реализация внешней сортировки

На вход этой операции подается неупорядоченный по множеству ключей K файл, то есть относительно этого множества ключей его можно рассматривать как множество однотипных записей. Относительно другого множества ключей это может быть упорядоченный файл. Как правило, объем исходного файла настолько велик, что невозможно ограничиться одним из алгоритмов внутренней сортировки и приходится использовать внешнюю сортировку. Наиболее популярный алгоритм внешней сортировки, основанный на методе слияния (балансного объединения) легко распараллеливается. На первом этапе этого алгоритма исходный файл делится на равные (за исключением последней) порции, которые сортируются в оперативной памяти параллельно работающих процессоров SP_1, …, SP_M. Для удобства можно считать $M=2^k$. Наименьшее значение M определяется отношением объема файла к объему оперативной памяти процессора, которую можно выделить для внутренней сортировки порции. С другой стороны, время сортировки в основном зависит от величины $\left(\frac{L}{M}\right)^2$ (L – количество записей в файле), что влечет за собой желание увеличить M настолько, насколько позволяет аппаратура. Но после первого этапа выполня-

ется k этапов слияния упорядоченных порций, на каждом из которых число порций уменьшается вдвое, а объем объединенных порций удваивается. Тогда вычислительная сложность рассматриваемого алгоритма будем иметь порядок, который определяется функцией $T(L,k)=\left(\frac{L}{2^k}\right)^2+2L\left(1-\frac{1}{2^k}\right)$. Их этого следует, что даже при значительном увеличении числа записей в файле величина k увеличивается незначительно. Например, при L=10^5 k=17, а при L=10^8 k=27. Однако даже при таких небольших значениях k число процессоров, используемых на первом этапе равное 2^k чрезвычайно велико, и значительно превосходит число процессоров (ядер) в современных суперкомпьютерах, и, тем более, в машинах баз данных, и в обычных корпоративных сетях. В некоторой степени выходом может служить использование графических процессоров, количество которых может быть большим и которые весьма эффективны при реализации алгоритмов, подобных внутренней сортировке. Был проведен анализ, суть которого состояла в оценке порядка вычислительной сложности операции сортировки при различных значениях k. Результаты анализа для различных значений k приведены в таблицах 4.1 и 4.2, а на рисунках 4.4 и 4.5 – их графическое представление.

Таблица 4.1. Оценки сложности параллельного алгоритма внешней сортировки при k =8 и k=10

L	$\min(T(L,k))$	$\frac{\min T(L,k)}{T(L,8)}$	$\frac{\min T(L,k)}{T(L,10)}$
100000	17	2,52	1,09
500000	19	8,63	1,48
1000000	20	16,25	1,95
1500000	21	23,88	2,43
3000000	22	46,77	3,86

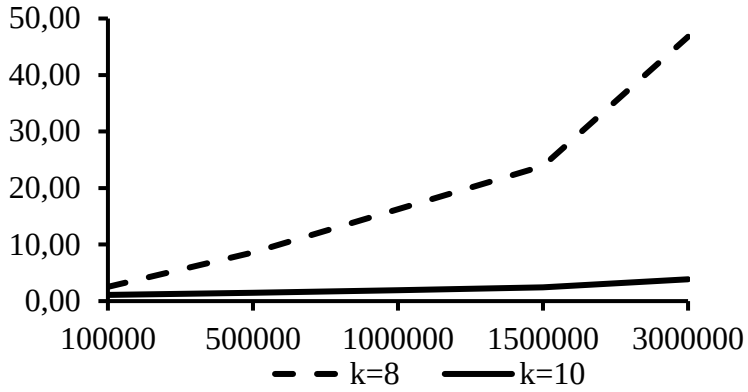
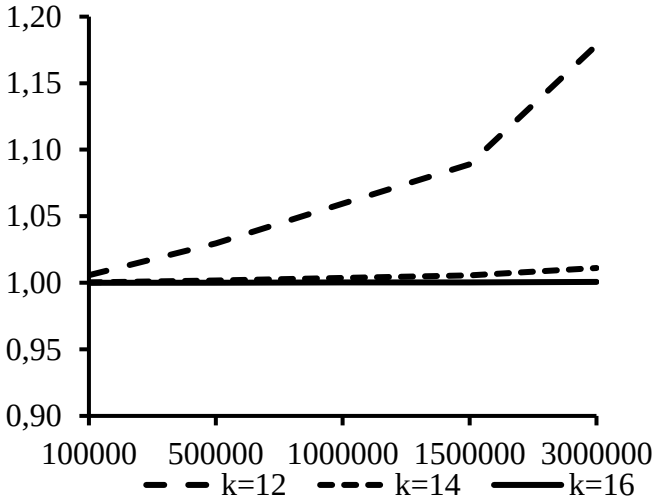


Рис. 4.4. Оценки сложности параллельного алгоритма внешней сортировки при k =8 и k=10

Таблица 4.2. Оценки сложности параллельного алгоритма внешней сортировки для k =12, k =14 и k=16

L	min $T(L, k)$	$\frac{\min T(L,k)}{T(L,12)}$	$\frac{\min T(L,k)}{T(L,14)}$	$\frac{\min T(L,k)}{T(L,16)}$
100000	17	1,01	1,00	1,00
500000	19	1,03	1,00	1,00
1000000	20	1,06	1,00	1,00
1500000	21	1,09	1,01	1,00
3000000	22	1,18	1,01	1,00



4.5. Оценки сложности параллельного алгоритма внешней сортировки для k =12, k =14 и k=16

Анализ показал, что при достаточно большом количестве процессоров (k =10, 12, 14, 16) отношение $\frac{\min T(L,k)}{T(L,k)}$ стремится к 1. Причем, при значительном

увеличении L это отношение мало изменяется. Например, при L=10^8 min($T(L$, $k)$)=27, и для k =16 $\frac{\min T(L,k)}{T(L,16)} = 1{,}02$.

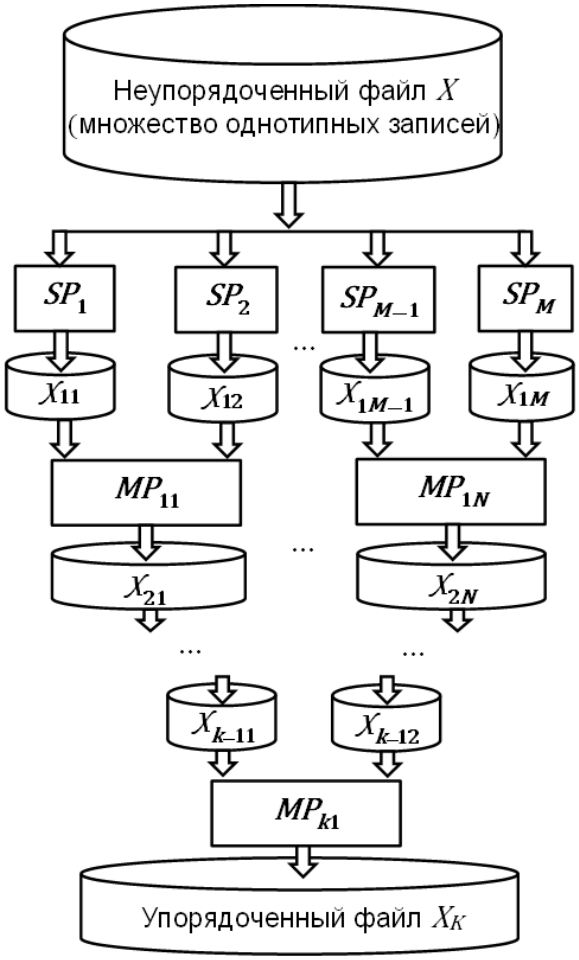


Рис.4.6. Архитектура программно-аппаратного комплекса для сортировки файла

Программно-аппаратный комплекс для реализации операции внешней сортировки имеет архитектуру, показанную на рисунке 4.6. На первом этапе M=2^k процессоров (SP_1, …, SP_M) выполняют программу внутренней сортировки файла X, разделенного на порции одинакового размера (за исключением, может быть, последней, если L не кратно M). В результате получается M упорядоченных файлов-порций X_{11}, X_{12}, …, X_{1M-1}, X_{1M}. Затем выполняется k этапов, на первом из которых $N = \frac{M}{2}$ процессоров (MP_1, …, MP_N) производят слияние пар, полученных на этапе разделения, в упорядоченные файлы-порции X_{11}, $X_{12} \rightarrow X_{21}$, …, X_{1M-1}, $X_{1M} \rightarrow X_{2N}$ удвоенного объема (в записях). На каждом последующем этапе используется вдвое меньше процессоров, а объемы получаемых в результате слияния упорядоченных файлов-порций удваиваются. Перед k-тым

этапом остаются два упорядоченных файла-порции $X_{k-1\,1}$ и $X_{k-1\,2}$, объем одного из них равен $\frac{L}{2}$, а объем второго может быть меньше, а в сумме их объемы равны L. На этом этапе используется единственный процессор MP_{k1}, который производит слияния двух последних упорядоченных файлов-порций в упорядоченный файл X_K.

4.3.2. Параллельная реализация операций выборки, слияния строго упорядоченных файлов и сечения

Операция выборки – это самая простая операция обработки файлов, она же и распараллеливается проще остальных операций. Как было показано в разделе 2.4.2 этой операции соответствует реляционная операция SELECT … WHERE …. Современные языки манипулирования данными в реляционных СУБД содержат средства, обеспечивающие автоматическое распараллеливание этой операции [198, 199]. Если использовать логически-последовательный метод доступа, то каждому из N процессоров можно поставить в соответствие фрагмент исходного файла. Объемы этих фрагментов, за исключением, может быть, последнего, равны $\frac{L}{N}$. Как и в случае первого этапа операции сортировки, объем последнего фрагмента может быть меньше этого значения. В большинстве СУБД разделение на фрагменты осуществляется на уровне индексных файлов. При последовательном доступе, если файл не разбит на фрагменты физически, каждому процессору ставится в соответствие номер первой записи фрагмента и объем фрагмента в записях. Каждый процессор выполняет выборку из своего фрагмента файла.

Операция слияния строго упорядоченных файлов требует использования метаданных (индексных файлов). В этом случае один из исходных строго упорядоченных файлов, как правило, имеющий больший объем, определяется как ведущий. Создается индексный файл, определяющий разбиение ведущего файла на равные (за исключением, может быть, последнего) фрагменты. Количество фрагментов равно количеству процессоров. Запись индексного файла для каждого фрагмента содержит адрес первой записи, значения ключей первой и

последней записей и объем этого фрагмента. На основе индексного файла ведущего файла создается индексный файл для ведомого файла. Естественно, фрагменты ведомого файла будут иметь различные объемы, но каждый из них будет содержать записи, значения ключей которых находятся в том же диапазоне, что и значения ключей соответствующего фрагмента ведущего файла. Следует отметить, что некоторые записи ведомого индексного файла будут содержать нулевое значение количества записей. В соответствии с определением файла из раздела 2.2.2 это означает, что все записи такого фрагмента ведомого файла есть универсальные неопределенные записи Θ, то есть фактически отсутствуют в ведомом файле. Однако в практике массовой обработки данных такие случаи встречаются очень редко. Для удобства эти индексные файлы могут быть объединены в один индексный файл, структура, которого приведена на рисунке 4.7.

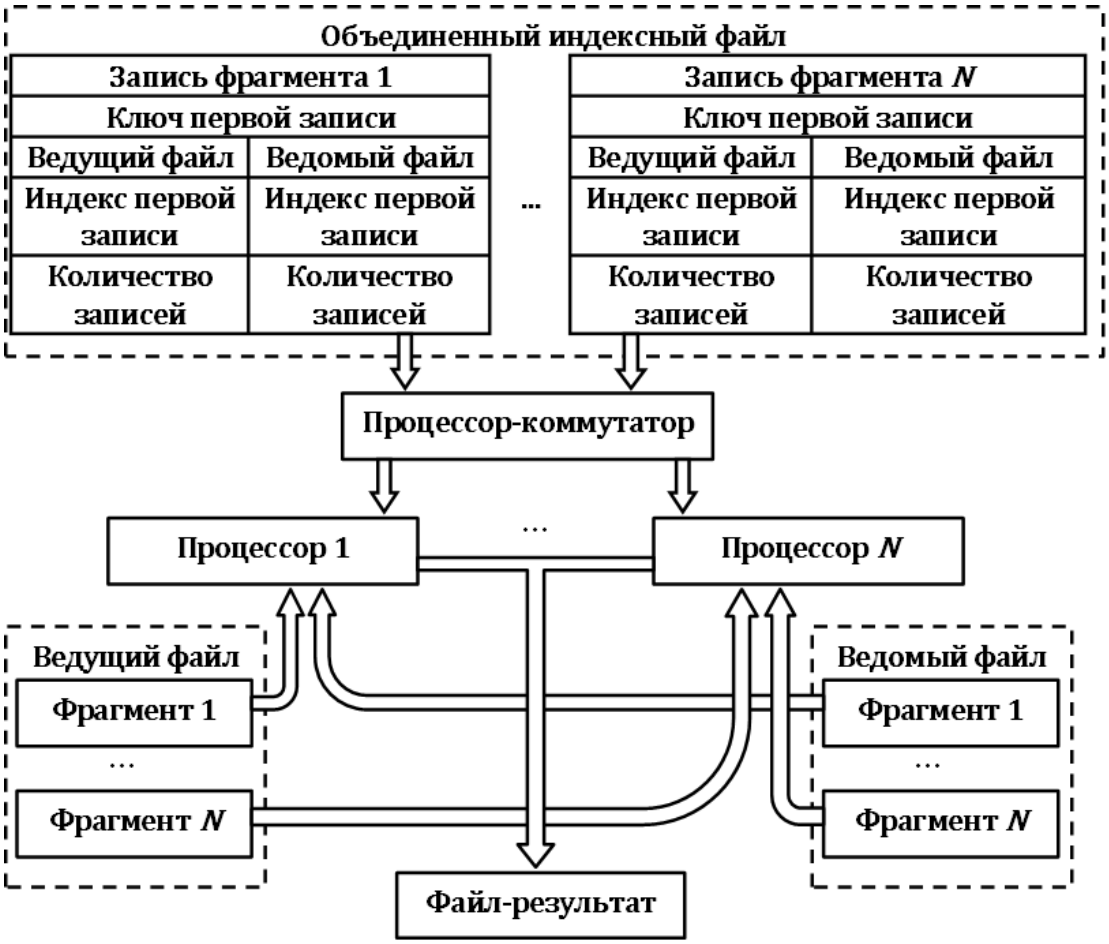


Рис. 4.7. Программно-аппаратный комплекс для слияния строго упорядоченных файлов

Программно-аппаратный комплекс для слияния строго упорядоченных файлов имеет архитектуру, показанную на рисунке 4.7. Процессор-коммутатор раздает N процессорам, соответствующие записи объединенного индексного файла. Процессоры выполняют программу слияния соответствующих им фрагментов исходных файлов в соответствии с алгоритмом, рассмотренным в разделе 2.2.3, формируя записи выходного файла.

Для реализации операции свертки необходимо использовать предложенный в разделе 3.5.2 принцип симметричного горизонтального распределения. В этом случае исходный файл разбивается на фрагменты программой, реализующей алгоритм распределения для одного файла. Тогда каждый класс эквивалентности целиком располагается в одном фрагменте, разница объемов фрагментов минимальна, и фрагменты сохраняют упорядоченность исходного файла. Количество фрагментов равно числу процессоров. Каждый процессор выполняет свертку классов эквивалентности своего фрагмента исходного файла в соответствии с алгоритмом, рассмотренным в соответствии с алгоритмом, рассмотренным в разделе 2.2.3, и формирует столько записей выходного файла, сколько классов эквивалентности в этом фрагменте.

4.4. Параллельная реализация операции слияния нестрого упорядоченных файлов в теоретико-множественной и реляционной моделях данных

Как было отмечено в разделе 3.3, операция слияния нестрого упорядоченных файлов относится к числу наиболее сложных операций МОД. Там же было отмечено, что разработчики СУБД используют различные методы для реализации реляционного аналога этой операции – операции Join. Для построения программно-аппаратных комплексов, которые реализуют операцию слияния нестрого упорядоченных файлов, в этом разделе будет использоваться метод, основанный на индексно-последовательном методе доступа и принципе сим-

метричного горизонтального распределения данных, описание которого приведено в разделе 3.5.2.

Для этого может быть использована любая параллельная вычислительная система класса SIMD, подобная машинам баз данных [89-92]. Особенность архитектуры таких систем заключается в том, что они состоят из хост-машины, реализующей общее управление вычислениями, и некоторого количества, как правило, однотипных процессоров (вычислителей), выполняющих одинаковые процедуры обработки данных. В некоторых реальных системах эти процессоры называются "лезвиями" (blades). Каждый из исходных файлов может храниться на одном внешнем накопителе, или его классы эквивалентности могут быть "разбросаны" по различным накопителям. Адрес каждого класса эквивалентности хранится в индексной части записи индексного файла. Хост-машина читает индексные файлы и сравнивает экземпляры множества ключей M^*. Индексы классов эквивалентности, экземпляры множества ключей которых совпали, передаются одному из вычислителей, параллельно реализующих декартово произведение. Процесс вычисления состоит из нескольких потоков (рисунок 4.8).

1. Потоки чтения файлов в память F_{11} и F_{12} независимо друг от друга считывают классы эквивалентности файлов в буферные зоны памяти. Адреса классов эквивалентности и количество записей в них получены от хост-машины.

2. Поток F_2 вычисляет декартово произведение классов эквивалентности X_{M^*} и Y_{M^*}. Этот поток может состоять из нескольких одинаковых параллельных нитей, каждая из которых соединяет одну запись класса эквивалентности X_{M^*} со всеми записями класса эквивалентности Y_{M^*}, помещая полученные записи в выходной буфер файла Z_M.

3. Поток вывода F_3 по мере поступления записей в выходной буфер перемещает их на внешний накопитель, на котором должен располагаться класс эквивалентности Z_{M^*}

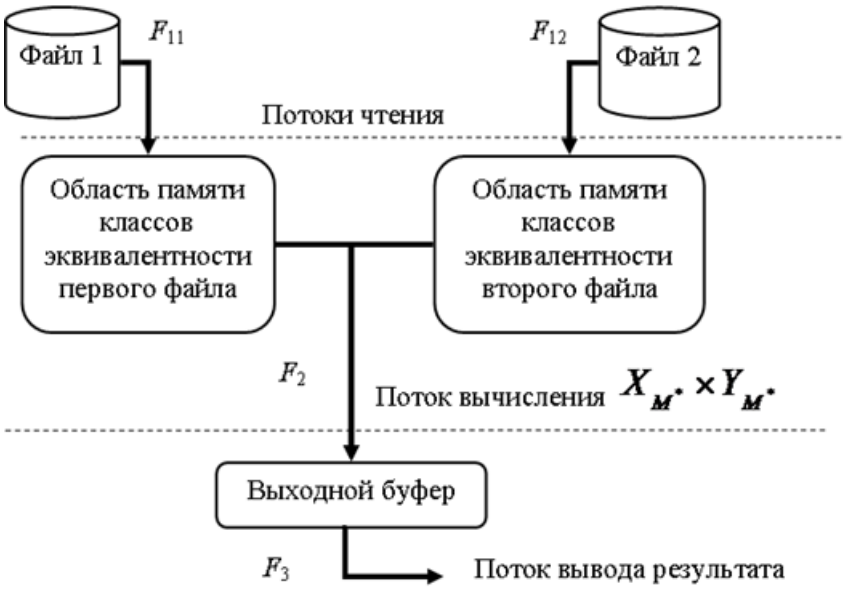


Рис. 4.8. Взаимодействие процессов в операции соединения

Очевидно, что предложенный подход позволяет использовать несколько уровней параллелизма.

Действительно, процесс чтения индексных файлов и подбора соответствующих классов эквивалентности осуществляется хост-машиной и не зависит от вычисления декартовых произведений на вычислителях. Поэтому чтение и сравнение индексных файлов осуществляется одновременно с вычислениями декартовых произведений. В реальных условиях количество классов эквивалентности может значительно превышать количество вычислителей. Информация о числе записей в классах эквивалентности, которая содержится в индексных файлах, позволяет распределить вычисления так, чтобы была обеспечена равномерная загрузка всей системы.

Процессы считывания обоих классов эквивалентности в буферные области оперативной памяти вычислителя могут происходить одновременно.

Вычисление декартова произведения также может быть распараллелено.

И, наконец, процесс вывода полученных записей в файл-результат осуществляется в асинхронном режиме, по мере их поступления в выходной буфер, и независимо от остальных процессов.

4.4.1. Параллельная реализация операции слияния нестрого упорядоченных файлов алгоритмом черпака

В основу этой реализации положен алгоритм 1, описание которого приведено в разделе 3.5.1. В этом случае класс эквивалентности ведущего файла целиком считывается (зачерпывается) в область оперативной памяти, которая называется черпак. Таким образом, черпак содержит все записи этого класса эквивалентности. В целях экономии оперативной памяти целесообразно экземпляр множества ключей K_i^* зачерпываемого класса эквивалентности сохранять в черпаке только один раз. Агрегаты неключевых полей записей этого класса эквивалентности образуют массив или список, длина которого определяется в процессе зачерпывания. Тогда у черпака будет следующая структура:

Экземпляр множества ключей K_i^*
Количество записей в классе эквивалентности $X_{K_i^*}$
Неключевые поля первой записи класса эквивалентности $X_{K_i^*}$
…
Неключевые поля последней записи класса эквивалентности $X_{K_i^*}$

Соответствующий экземпляру множества ключей K_i^* класс эквивалентности ведомого файла считывается по одной записи. Каждая прочитанная запись выходного файла обрабатывается совместно со всеми записями черпака, и таким образом формируется декартово произведение этих классов эквивалентности. Этот алгоритм можно легко распараллелить, поскольку и файлы, и черпак, реализованный как массив или список, обладают параллелизмом данных. Причем, распараллеливание можно провести на двух уровнях.

Первый уровень распараллеливания основан на принципе симметричного горизонтального распределения. Исходные файлы разбиваются на фрагменты, которые содержат классы эквивалентности, соответствующие одним и тем же экземплярам множества ключей. То есть существует взаимно-однозначное соответствие между этими фрагментами (фактор-множествами), при котором каждому классу эквивалентности фрагмента файла X_K взаимно-однозначно со-

ответствует класс эквивалентности фрагмента файла Y_K. Если X_K^i и Y_K^i – соответствующие друг другу фрагменты исходных файлов X_K и Y_K, то их классы эквивалентности соответствуют друг другу так, как это показано на рисунке 4.9.

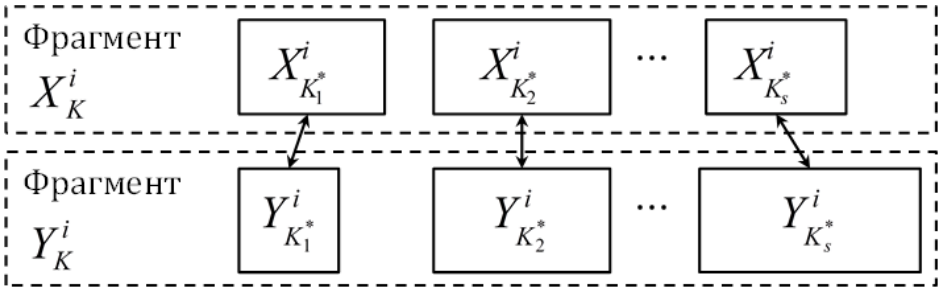


Рис. 4.9. Соответствие классов эквивалентности фрагментов файлов

Для каждой пары фрагментов исходных файлов выделяется свой фрагмент-процессор, который выполняет декартовы произведения классов эквивалентности и формирует записи выходного файла.

Второй уровень распараллеливания обусловлен тем, что с каждой записью ведомого файла обрабатываются все, хранящиеся в черпаке, записи ведущего файла. Тогда, если имеется некоторое количество процессоров, черпак делится на такое же количество частей. Каждый из процессоров выполняет декартово произведение своей части черпака и класса эквивалентности ведомого файла. В дальнейшем эти процессоры называются ДП-процессорами.

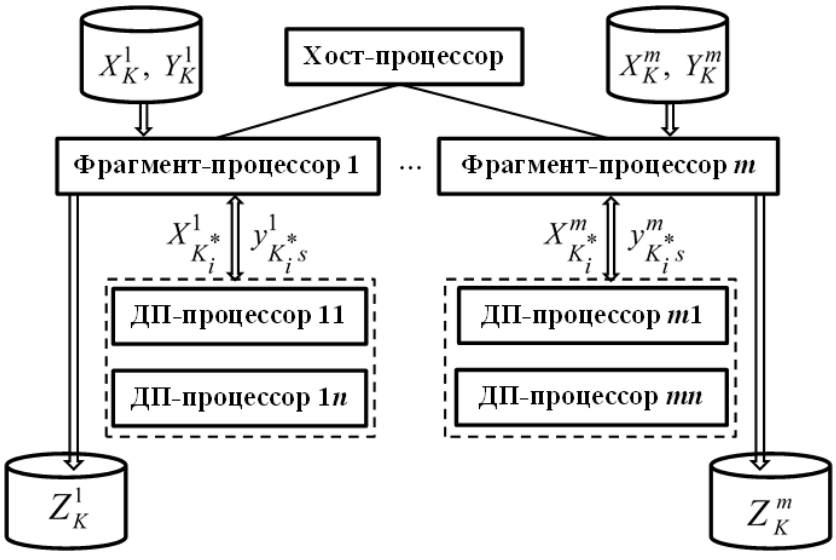


Рис. 4.10. Архитектура программно-аппаратного комплекса для слияния нестрого упорядоченных файлов

Из сказанного следует, что программно-аппаратный комплекс для реализации операции слияния нестрого упорядоченных файлов имеет архитектуру, приведенную на рисунке 4.10.

Хост-процессор, связанный с приложениями пользователей, осуществляет общее управление всем комплексом при решении прикладных задач. Для каждой базы данных, файлы которой симметрично горизонтально распределены, имеется карта распределения этих файлов по массовым запоминающим устройствам. Хост-процессор запускает обработку фрагментов файлов на фрагмент-процессорах, передавая каждому из них данные о расположении фрагментов файлов. Фрагмент-процессор последовательно зачерпывает классы эквивалентности ведущего файла. Черпак делится на равные, исключая, быть может, последнюю, части, начальные адреса и объемы (число записей) которых передаются соответствующим ДП-процессорам. Для каждого расположенного в черпаке класса эквивалентности фрагмент-процессор последовательно читает по одной записи соответствующего класса эквивалентности ведомого файла. Этот процесс взаимодействия фрагмент процессора с ДП-процессорами показан на рисунке 4.11.

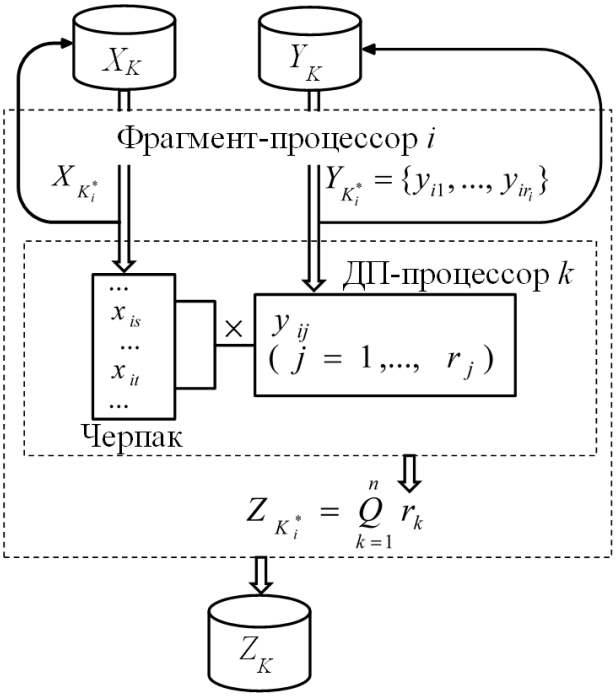


Рис. 4.11. Взаимодействие фрагмент-процессора с ДП-процессорами

ДП-процессор поочередно обрабатывает все записи класса эквивалентности ведомого файла со всеми записями черпака, формируя при этом последовательность записей выходного файла (r_1, …, r_n), где $n = L(X_K^i) \times L(Y_K^i)$ – произведение объемов i-тых классов эквивалентности исходных файлов. Эти записи передаются фрагмент-процессору, который формирует из них класс эквивалентности выходного файла. Формирование класса эквивалентности выходного файла, как правило, осуществляется при помощи некоторой групповой операции (операции квантификации), например, операции суммирования, вычисления среднего значения, поиск минимального или максимального значения и тому подобных операций. То есть, в процессе формирования класса эквивалентности выходного файла обычно выполняется и операция свертки. Таким образом, после обработки черпака со всеми записями класса эквивалентности ведомого файла, формируется класс эквивалентности файла-результата.

Рассматриваемому варианту МОД присущи регулярные запросы. Такие запросы разрабатываются прикладным программистом в процессе проектирования автоматизированной информационной системы, после чего существуют практически без изменений на протяжении всего жизненного цикла системы. Следовательно, затраты на трансляцию запроса, его оптимизацию, создание библиотечной программы, аналогичной хранимой процедуре, имеют разовый характер и не наносят ущерба как разработчику – профессиональному прикладному программисту, – так и самой информационной системе во время ее эксплуатации.

Особенность современного состояния вычислительной техники состоит в том, что, во-первых, используются гибкие архитектуры современных аппаратных средств, основанные на многоядерности и многопроцессорности, и, во-вторых, вычислительные сети обеспечивают такую простоту коммуникаций, которая позволяет легко проектировать различные топологии сетей. Эти два фактора – регулярность запросов и гибкость вычислительных средств – позволяют решать задачи оптимизации архитектуры программно-аппаратного вы-

числительного комплекса на этапе разработки автоматизированной информационной системы. Причем для каждой системы, входящей в ее состав задачи или даже отдельного запроса может быть разработан индивидуальный программно-аппаратный вычислительный комплекс.

Таким образом, в задачах МОД можно выделить две основные цели оптимизации: ускорение отдельных операций обработки одного или нескольких файлов, и построение оптимальных последовательностей операций (процессов или запросов) для решения конкретных задач.

На основе файловой модели становится возможной оптимизация отдельных операций за счет применения различных параллельных архитектур вычислительных комплексов, реализующих выполнение операций. То есть, она позволяет производить двухуровневую оптимизацию, как на уровне операций, так и на уровне процессов. Наибольшую сложность при решении проблемы параллельной обработки больших объемов данных имеют операция слияния нестрого упорядоченных файлов и процессы, состоящие из последовательностей таких операций.

4.4.2. Параллельная реализация последовательности операций слияния нестрого упорядоченных файлов

Далее рассматривается построение программно-аппаратного комплекса, который может эффективно решать различные задачи МОД. В терминах файловой модели данных алгебраической моделью процесса МОД служит алгебраическое выражение вида $A = E(A_1, \ldots, A_p)$, правая часть которого состоит из файлов $A_1, \ldots, A_p$, соединенных знаками операций над файлами, а левая – из выходного файла. Далее рассматриваются только аддитивная операция слияния строго упорядоченных файлов и мультипликативная операция слияния нестрого упорядоченных файлов. Тогда правая часть алгебраического выражения $E(A_1, \ldots, A_p)$ может быть сведена к "сумме произведений" исходных файлов. Как было сказано, в реляционной модели данных операции слияния нестрого упорядоченных файлов соответствует операция JOIN. Далее рассмотрена архитектура вычислительного комплекса для параллельного вычисления цепочки

"произведений" файлов, то есть, в терминах SQL-реляционной модели данных, вычисления выражения запроса вида:

SELECT <список полей> FROM

(A_p INNER JOIN SELECT <список полей> FROM

(A_{p-1} INNER JOIN …

SELECT <список полей> FROM

(A_2 INNER JOIN A_1 ON π_1) …) ON π_{p-1})

Для вычисления этого запроса может быть эффективно использована конвейерная (MISD) архитектура вычислительного комплекса (рисунок 4.11).

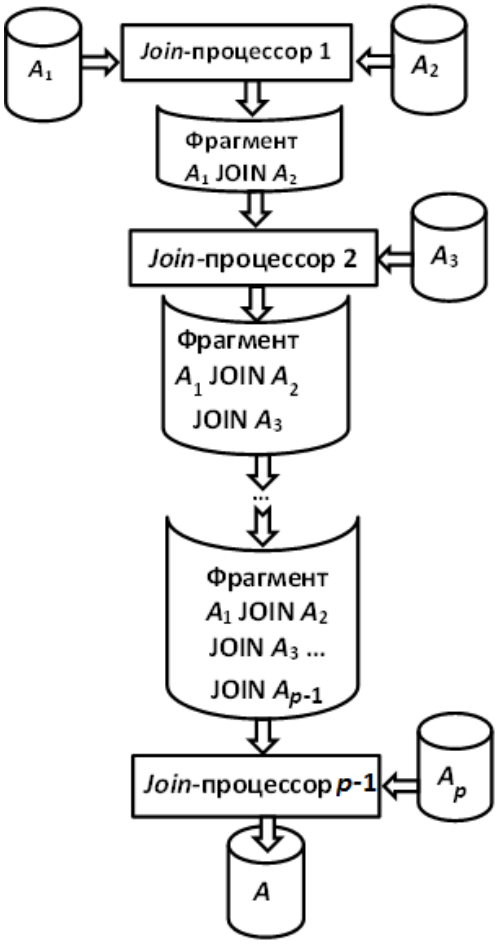


Рис.4.11. Конвейерный вычислительный комплекс для вычисления цепочки операций Join

Этот комплекс состоит из $p-1$ *Join*-процессора, первый из которых начинает обработку первых двух "сомножителей". Как только готов очередной фрагмент (один или несколько классов эквивалентности) результата, он передается

следующему *Join*-процессору, который обрабатывает этот фрагмент с соответствующим классом эквивалентности файла A_3, и так далее по конвейеру. Последний *Join*-процессор конвейера принимает классы эквивалентности, полученные в результате предыдущих произведений и обрабатывает их с соответствующими классами эквивалентности последней таблицы в цепочке.

Среди операций МОД операция JOIN имеет наибольшую вычислительную сложность. На основе файловой модели *Join*-процессор можно организовать как вычислительный комплекс на основе SIMD-архитектуры (рисунок 4.12).

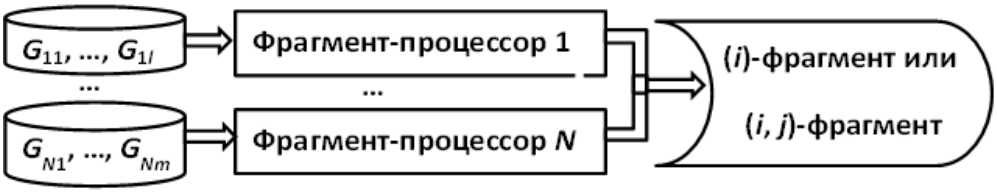


Рис. 4.12. SIMD-архитектура Join-процессора

Фрагменты таблиц-операндов, обрабатываемых *i*-тым фрагмент-процессором, можно рассматривать как совокупность пар G_{i1}, …, G_{ik}, каждая из которых содержит два соответствующих друг другу класса эквивалентности таблиц-операндов. Размещение данных в памяти фрагмент-процессоров производится на основе принципа симметричного горизонтального распределения.

Фрагмент-процессор выполняет операцию JOIN над фрагментами файлов и передает на выход либо весь результат операции, либо фрагмент, содержащий результат декартова произведения классов эквивалентности очередной пары G_{ij}.

При таком сочетании конвейерной (MISD) и SIMD-архитектур вычислительных комплексов можно добиться значительного повышения эффективности процессов МОД.

Такой подход позволяет использовать несколько различных потоков. Потоки чтения индексно-последовательных файлов читают значения ключей в области памяти. Далее поток распределения передает значения ключей и расположения классов эквивалентности кластеру вычислителей. Кластер, состоящий из нескольких вычислителей, выполняет декартово произведение классов экви-

валентности. Кластер в данном случае может быть, как машиной баз данных, так и группой компьютеров, объединенных сетью под общим управлением.

4.4.3. Параллельная реализация операции слияния нестрого упорядоченных файлов с использованием ассоциативных вычислительных систем

Вычислительная система с ассоциативным распределением ресурсов (ВСАРР) создавалась для решения широкого класса математических задач. Академик Бурцев В.С. инициировал исследование данной архитектуры для решения задач нечисловой обработки информации. Параллельная система баз данных на основе вычислительной системы «ВСАРР» работает с базами данных, обеспечивая аппаратную поддержку ассоциативного поиска. Использование вычислительной системы «ВСАРР», работающей по принципу потока данных, помогает решить ряд проблем, которые возникали при использовании универсального аппаратного обеспечения.

В архитектуре «ВСАРР» исполнение программы описывается в терминах приема, обработки и выдачи токенов, содержащих некоторое данное и тег. Зависимости между данными транслируются в совпадение и преобразование тегов, в то время как обработка данных имеет место тогда, когда несколько совпавших токенов приходят в исполнительное устройство. Команда, которая должна быть выбрана из памяти команд (в соответствии с информацией, присутствующей в теге токена), содержит информацию о том, что должно быть выполнено над пришедшими данными и как следует изменить теги. Устройство совпадения и исполнительное устройство соединяются через асинхронный конвейер с очередями, добавленными для сглаживания неравномерностей нагрузки. Поиск токенов с совпадающими тегами выполняется в ассоциативной памяти.

Процесс вычислений наглядно изображается в виде направленного графа, по дугам которого перемещаются тегированные данные (токены), а в именованных узлах которого исполняется одна или несколько команд. Каждый узел

графа имеет один или два входа и много выходов. Узел представляет собой одну операцию или несколько операций, называемых программой узла [200, 201].

Граф вычислений [202], приведенный на рисунке 4.13, соответствует операции соединения без использования индексно-последовательных файлов.

Память команд исполнительного устройства:

Адрес программы узла	**Команда 1**	**Команда 2**	**Команда 3**
1	*ЧТЕНИЕ*	*Контекст=0*	Выдать токен <Адрес программы узла 2>
2	ДЕКАРТОВО ПРОИЗВЕДЕНИЕ	Выдать токен результат <Адрес результата>	

Граф алгоритма имеет четыре узла: два экземпляра узла «чтения» для каждого из файлов, один экземпляр узла «сравнение» и один экземпляр узла «декартово произведение» (рисунок 4.13). На входы узлов поступают токены: тегированные данные, в тегах которых передается ключ поиска в ассоциативной памяти: физический адрес программы узла в памяти команд и контекст исполнения программы узла.

Узел «чтение» получает на вход адрес нестрого упорядоченного файла, токены будут идентифицироваться по значению ключа класса эквивалентности, узел «декартово произведение» будет выполнять декартово произведение поступивших в него данных. Сравнение классов эквивалентности будет происходить в ассоциативной памяти.

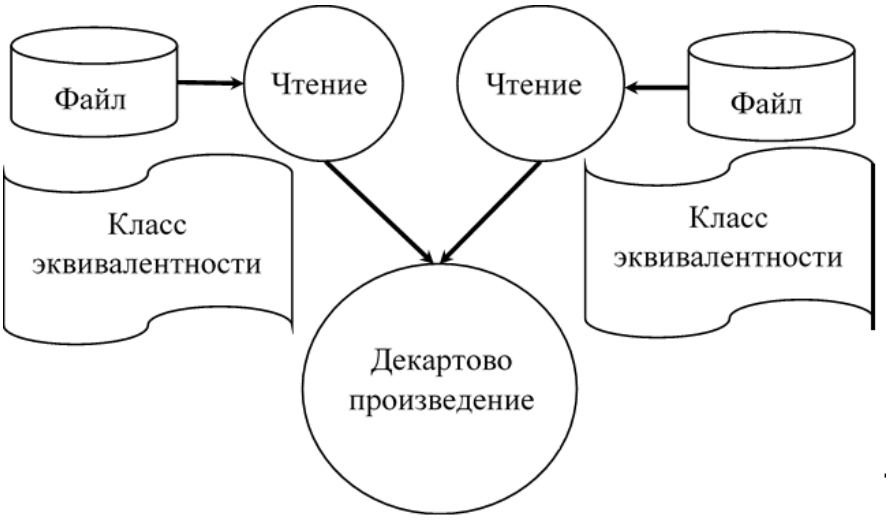


Рисунок 4.13. Граф вычислений операции соединения.

При использовании файла индексов граф вычислений имеет два узла чтения индексных файлов («чтение иф»), узел распределения, два узла чтения, два узла «чтение» (выполняющие чтения нестрого упорядоченных файлов), узел декартово произведения. Граф вычислений изображен на рисунке 4.14.

Адрес программы узла	**Команда 1**	**Команда 2**	**Команда 3**
1	*ЧТЕНИЕ ИФ*	*Контекст=0*	Выдать токен <Адрес программы узла 2>
2	*РАСПРЕДЕЛЕНИЕ*	*Контекст=0*	Выдать токен <Адрес программы узла 3>
3	*ЧТЕНИЕ*	*Контекст=0*	Выдать токен <Адрес программы узла 4>
4	ДЕКАРТОВО ПРОИЗВЕДЕНИЕ	Выдать токен результат <Адрес результата>	

Узел «чтение иф» выполняет чтение файлов индексов и передает их в узел распределения. Узел распределения передает в узлы «чтение» индекс и адреса и мощности классов эквивалентности. Узлы «чтения» передают про-

читанные данные узлу «декартово произведение», который в свою очередь передает результат на вывод.

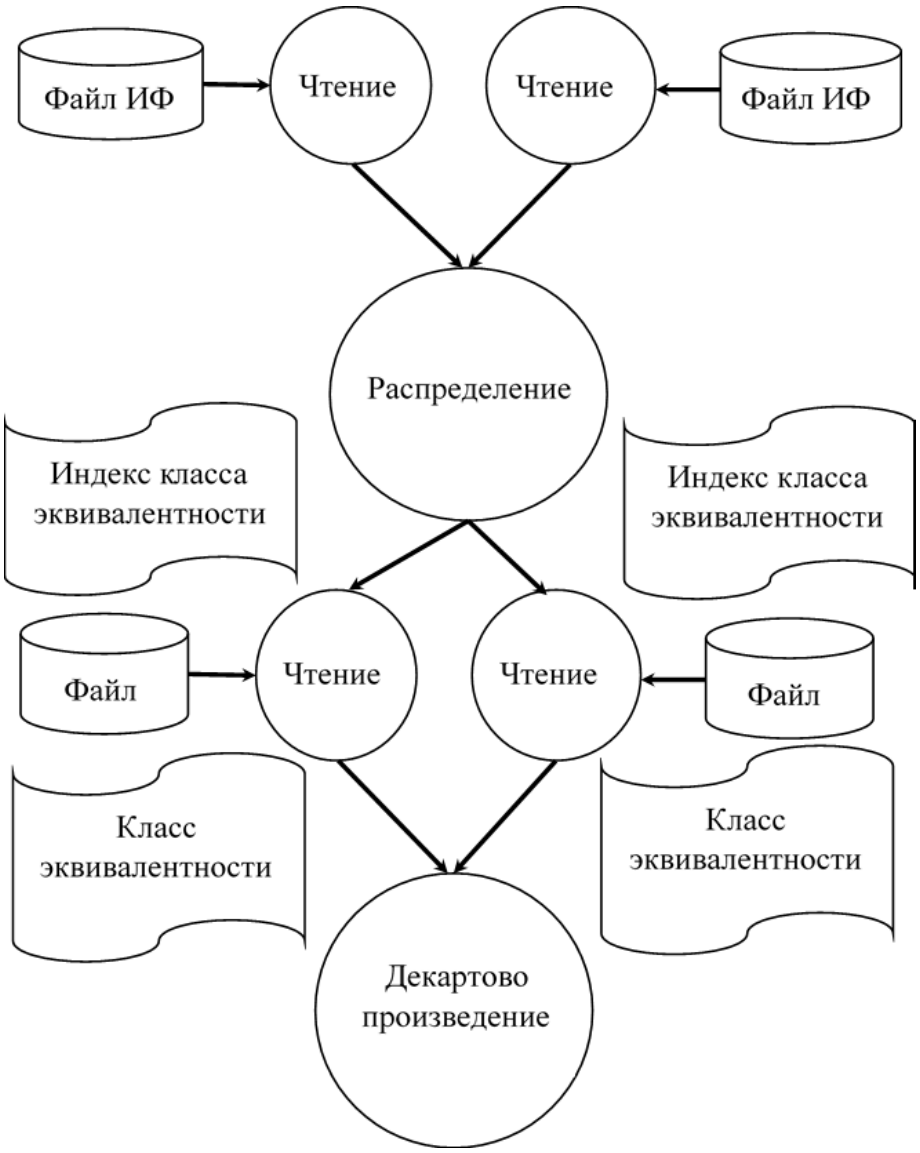


Рисунок 4.14. Граф вычислений операции соединения
с использованием файла индексов

4.5. Заключительные замечания к главе 4

В главе рассмотрены этапы построения программно-аппаратных комплексов, реализующих МОД.

Предложена архитектура программно-аппаратного комплекса для реализации многомерно-матричной модели данных. Подробно рассмотрена архитектура, ориентированная на реализацию операции (λ, μ)-свернутого произведения многомерных матриц на основе сечений по совокупности скоттовых индексов.

Рассмотрены возможные архитектуры программно-аппаратных комплексов для реализации простых операций теоретико-множественной модели данных: внешней сортировки, выборки, сечения и слияния строго упорядоченных файлов.

Предложена архитектура программно-аппаратного комплекса для параллельной реализации операции слияния нестрого упорядоченных файлов в теоретико-множественной и реляционной моделях данных на основе принципа симметричного горизонтального распределения таблиц-операндов и приводится описание параллельной реализации этой операции для выскоактивных данных с использованием алгоритма черпака.

Предложена и рассмотрена параллельная конвейерная реализация последовательности операций слияния нестрого упорядоченных файлов на основе стандартных архитектур вычислительных систем.

Предложена параллельная реализация операции слияния нестрого упорядоченных файлов на основе архитектуры вычислительных систем с ассоциативным распределением ресурсов.

Основные результаты, полученные в данной главе, были опубликованы в работах [117, 188-192, 202].

Глава 5. ЭКСПЕРИМЕНТАЛЬНЫЙ АНАЛИЗ ПРОГРАММНО-АППАРАТНЫХ РЕАЛИЗАЦИЙ МАССОВОЙ ОБРАБОТКИ ДАННЫХ

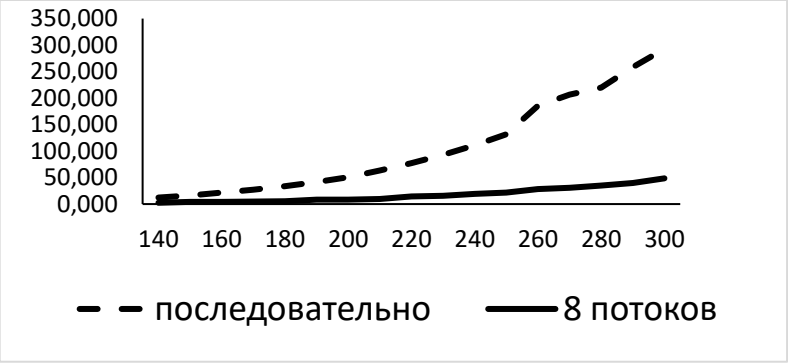
5.1. Анализ параллельной реализации операции умножения многомерных матриц

Для умножения использовался блочный параллельный алгоритм умножения многомерных матриц, описание которого приведено в главе 3.

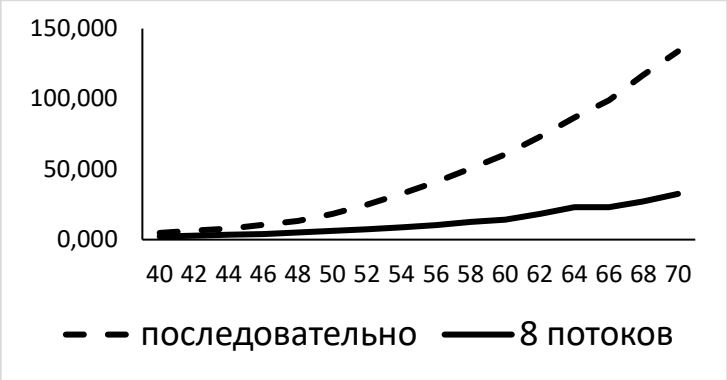
Эксперимент [147, 174, 175] проводился для трехмерных (A_{lsc}, B_{scm}) и четырехмерных ($A_{ls_1s_2c}, B_{s_1s_2cm}$) числовых матриц. Все индексы трехмерных матриц принимали значения от 10 до 300 с шагом 10, а четырехмерных – 10 до 70 с шагом 2.

На каждом шаге эксперимента выполнялись последовательное и параллельное (8 и 27 потоков) умножение многомерных матриц. Таким образом, для трехмерных матриц выполнялось (1, 1)-свернутое произведение, а для четырехмерных – (2, 1)- свернутое произведение. Программно-аппаратные комплексы создавались как виртуальные машины Microsoft Azure на основе процессора Xeon E5-2673. Использовались конфигурации с восьмью и шестнадцатью ядрами.

Результаты эксперимента, приведенные на рисунке 5.1, показывают, что с увеличением размерности индексов возрастает эффективность параллельного алгоритма.



А) трехмерные матрицы



Б) четырехмерные матрицы

Рис. 5.1. Зависимость производительности алгоритма от размерности индексов

На рисунке 5.2 приведены результаты анализа производительности алгоритма параллельного умножения многомерных матриц в зависимости от соотношения числа потоков и числа вычислителей (ядер).

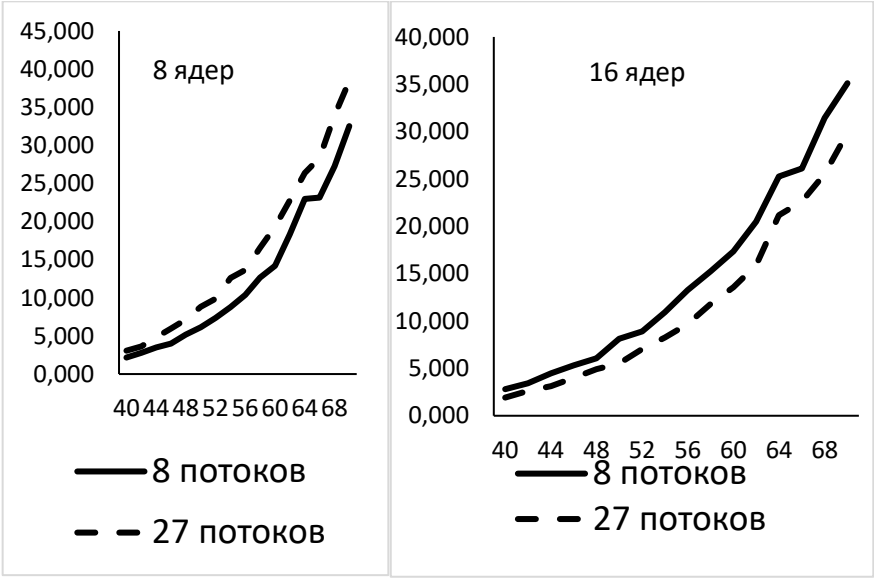


Рис. 5.2. Зависимость производительности алгоритма от числа вычислителей

Результаты эксперимента позволяют утверждать, что предложенный алгоритм обеспечивает высокую производительность при умножении многомерных матриц.

5.2. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов

Для параллельной реализации теоретико-множественной модели данных могут использоваться различные архитектуры вычислительных комплексов, например, массивно-параллельные системы (MPP) или симметричные мультипроцессорные системы (SMP). То есть, это могут быть как неоднородные, так и однородные вычислительные системы, в простейшем случае – система с одним или несколькими многоядерными процессорами. Единственное требование состоит в том, чтобы каждый процессор (ядро) был ассоциирован с собственным устройством массовой памяти (в простейшем случае – дисковым накопителем). Общая архитектура вычислительного комплекса для реализации теоретико-множественной модели приведена на рисунке 5.3.

Рис. 5.3. Архитектура вычислительной системы для реализации теоретико-множественной модели

5.3. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов средствами СУБД и SMP-архитектуры

Для оценки качества распараллеливания операции слияния нестрого упорядоченных файлов на симметричной мультипроцессорной системе был проведен вычислительный эксперимент на основе вычислительной системы со следующими характеристиками:

- четырехядерный центральный процессор Intel CORE i7, поддерживающий технологию гиперпоточности (HTT) и, соответственно, имеющий восемь виртуальных ядер;
- оперативная память 6 Гбайт;
- внешняя память 1 Тбайт.

Исходные данные представляли собой таблицы базы данных Microsoft Sql-Server, а операция слияния нестрого упорядоченных файлов реализовывалась операцией Join. Подфайлы-операнды каждой пары генерировались таким образом, что содержали классы эквивалентности с одинаковыми значениями экземпляров множества ключей, но различными количествами записей. На каждом шаге генерировалось по четыре пары подфайлов, которые одновременно обрабатывались четырьмя параллельными процессами, результаты которых объединялись в один файл-результат. Затем все подфайлы-операнды объединялись в два файла операнда. И выполнялся один процесс, реализовавший операцию слияния нестрого упорядоченных файлов последовательно. Количество обрабатываемых на каждом этапе записей приведено в таблице 5.1.

Таблица 5.1. Объемы данных на каждом шаге эксперимента

Шаг	Количество записей (тысячи)	
	Подфайлы пар (X_M, Y_M)	Файлы (X_M, Y_M)
1	200	400
2	400	800
3	600	1200
4	800	1600
5	1200	2400

В ходе эксперимента были получены приведенные на рисунке 5.4 соотношения между объемами базы данных (в тысячах записей) и временами обработки (в секундах) этой базы четырьмя параллельными процессами и одним последовательным процессом. На графике видно, что параллельная реализация операции слияния нестрого упорядоченных файлов четырьмя процессами требует в среднем в пять раз меньше времени, чем последовательная реализация.

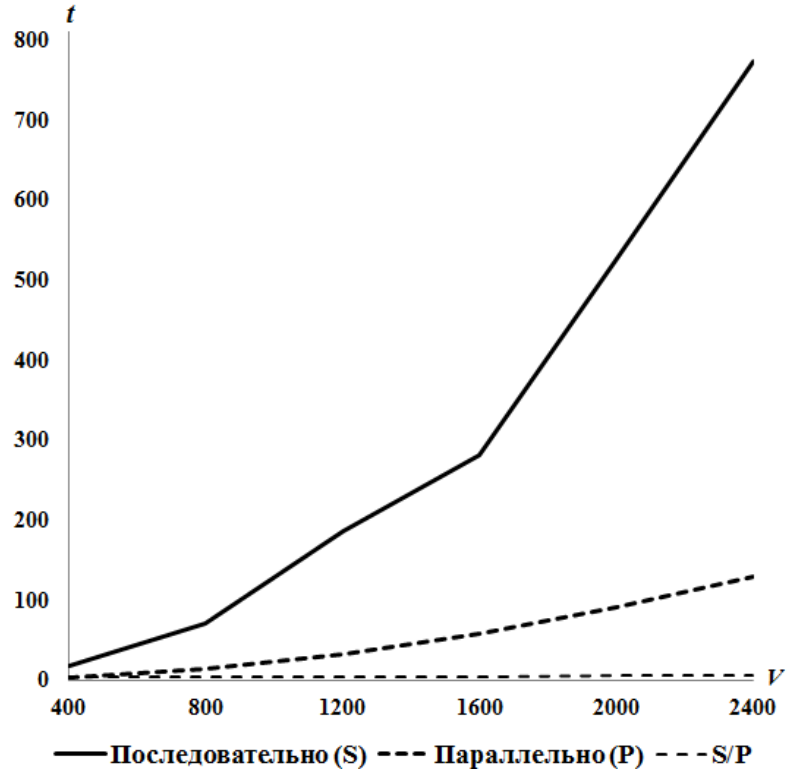


Рис. 5.4. Зависимость времени выполнения операции слияния от объема данных

Анализ загрузки процессами ресурсов (виртуальных ядер) центрального процессора показал (рисунок 5.5), что каждый процесс выполняется на одном виртуальном ядре, не более чем на половину используя его ресурсы. Из этого следует, что число параллельно выполняющихся процессов может быть вдвое большим, чем число выделенных для реализации операции ядер.

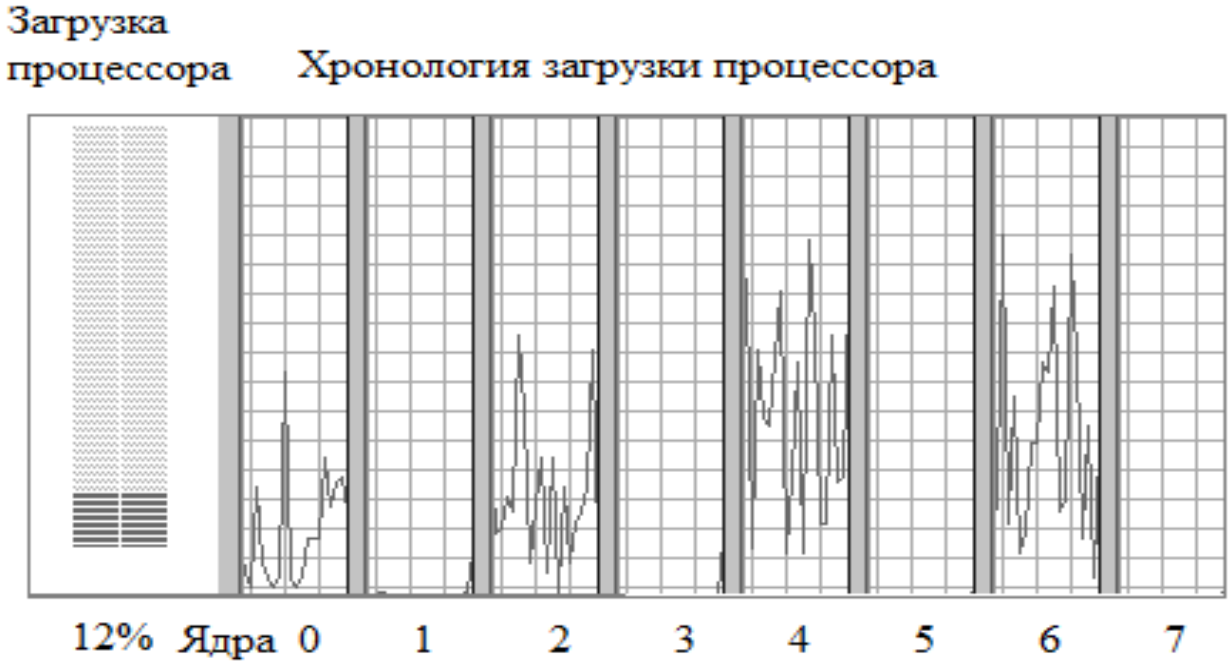


Рис. 5.5. График загрузки ресурсов процессора

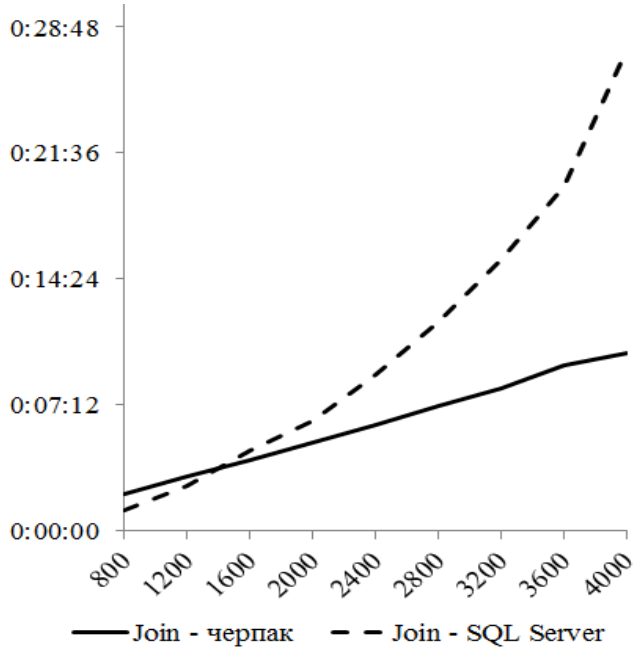


Рис. 5.6. Время реализации операции слияния нестрого упорядоченных файлов (Join)

Результаты эксперимента, приведенные на рисунке 5.6, подтверждают, что применение предложенного метода параллельной реализации операции слияния нестрого упорядоченных файлов более эффективно, чем реализация стандартными средствами распараллеливания операции Join, реализованными в СУБД.

При реализации операции слияния нестрого упорядоченных файлов алгоритмом черпака загрузка ядер процессора оказалась весьма высокой, что показано на рисунке 5.7.

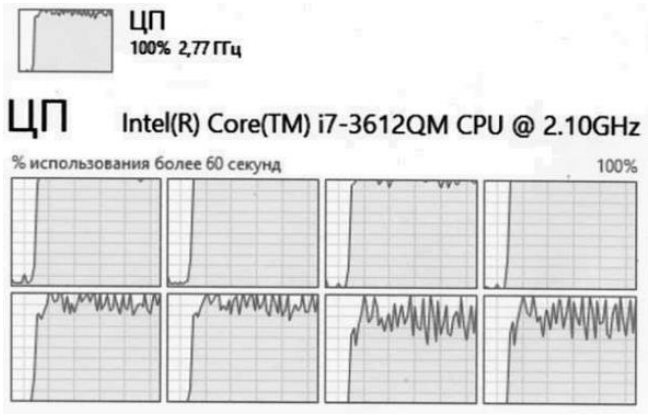


Рис.5.7. Загрузка процессоров (ядер) в виртуальном программно-аппаратном комплексе

5.4. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием MPP-архитектуры в облачной среде

Облачные вычисления определяется как вычислительные концепции, которые подразумевают большое количество компьютеров, подключенных через сеть реального времени, подобную Интернет. Это синоним для распределенных вычислений по сети, а значит, возможность запуска программы или приложения на многих подключенных компьютерах одновременно. Облачные вычисления основаны на данных, приложениях, инфраструктуре и различных платформах для разработчиков приложений, расположенных в глобальной сети (WAN).

Облачные системы предоставляют программисту большой набор инструментальных средств разработки приложений. Они объединены в подсистему, которая называется Платформа как сервис (Platform as a Service – PaaS). PaaS – это предоставление облачных вычислений, при котором потребитель получает доступ к использованию технологических платформ, размещённых в облачной системе, таких как: операционные системы, системы управления базами данных, связующее программное обеспечение, средства разработки и тестирования приложений. Далее рассматриваются методы и ресурсы необходимые для параллельной обработки распределенных данных, которые обеспечивает разработчикам PaaS облачной системы Windows Azure.

Первый основной ресурс – это виртуальная машина. Виртуальная машина конструируется пользователем в соответствии с его требованиями. Минимальные возможности – это одноядерный процессор с 768 Мбайтами памяти, а наиболее мощный вариант – восьмиядерный процессор с 56 Гбайтами памяти. На виртуальной машине устанавливается операционная система Windows Server. Если виртуальная машина используется для работы с базами данных, то на ней также устанавливается СУБД Microsoft SQL Server Azure. Версия и настройка программного обеспечения определяется мощностью машины.

Второй основной ресурс – это отдельная база данных. Программисту предоставляется возможность создания необходимого количества баз данных на облачных ресурсах. Эти БД могут быть физически независимыми друг от друга, то есть располагаться на различных облачных хранилищах данных и обрабатываться различными облачными процессорами. Для создания БД и выполнения запросов к ней используется СУБД Microsoft SQL Server Azure.

Третий ресурс, который может быть использован для решения вспомогательных задач параллельной обработки распределенных данных, – хранилище таблиц. К таблицам можно применять только простые запросы, не содержащие бинарных операций типа операции соединения. Таблицы могут использоваться как индексные файлы при индексно-последовательной организации данных.

Далее рассматриваются методы взаимодействия прикладных программ с этими основными ресурсами [145, 193]. При этом предполагается, что:

– организация данных реализована на основе принципа симметричного горизонтального распределения;

– программа реализует параллельное выполнение независимых потоков, которые вместе реализуют массовую обработку данных.

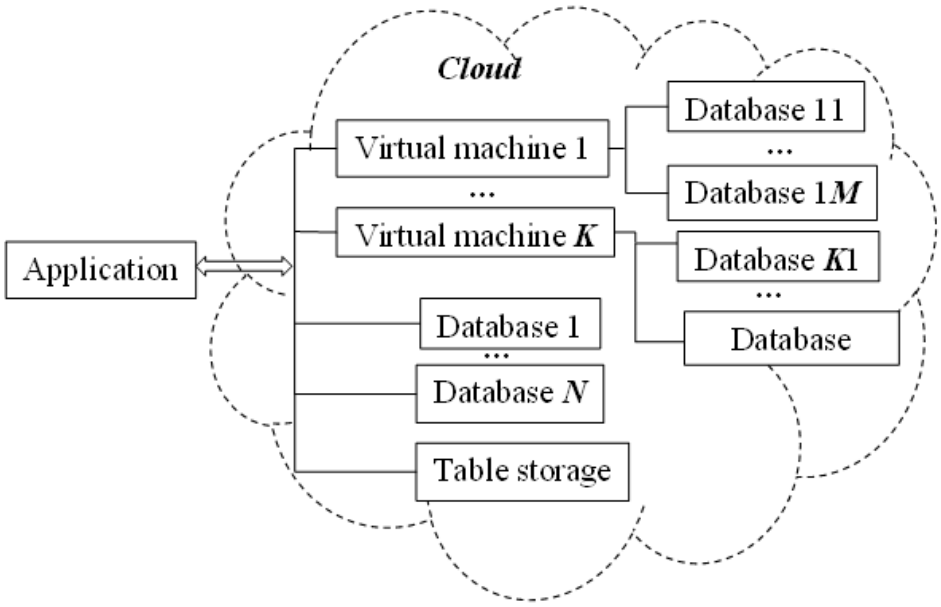


Рис. 5.8. Взаимодействие между прикладной программой и облачными ресурсами

Общая схема взаимодействия прикладной программы с основными облачными ресурсами показана на рисунке 5.8.

Прикладная программа, которая реализует массовую обработку данных, может располагаться на рабочей станции пользователя или на виртуальной машине в облаке. В последнем случае пользователь получает к ней доступ с помощью удаленного рабочего стола.

Обработка данных, расположенных в облаке, может быть реализована различными способами.

Первый способ реализует параллельную обработку на уровне запросов. В этом случае алгебраическое выражение запроса разбивается на подвыражения, которые могут выполняться одновременно на разных процессорах или ядрах одного многоядерного процессора.

Пример 1. Пусть даны две таблицы с одинаковыми схемами: P(A, B) и Q(A, B), и таблица R(A, C). Эти таблицы участвуют в запросе, в котором сначала выполняется операция объединения таблиц P и Q, а затем операция JOIN над результатом объединения и таблицей R. Этот запрос имеет вид:

SELECT R.A, PUQ.B, R.C FROM R INNER JOIN PUQ ON R.A =PUQ.A;

PUQ – запрос на объединение таблиц P и Q:

SELECT P.A, P.B FROM P UNION SELECT Q.A, Q.B FROM Q;

Этот запрос может быть преобразован в эквивалентный запрос:

SELECT R.A, P.B, R.C FROM R INNER JOIN P ON R.A = P.A

UNION

SELECT R.A, Q.B, R.C FROM R INNER JOIN Q ON R.A = Q.A;

(операции JOIN и UNION подчиняются закону дистрибутивности).

Схема взаимодействия прикладной программы с распределенными между двумя БД в облаке данными и запросами приведена на рисунке 5.9.

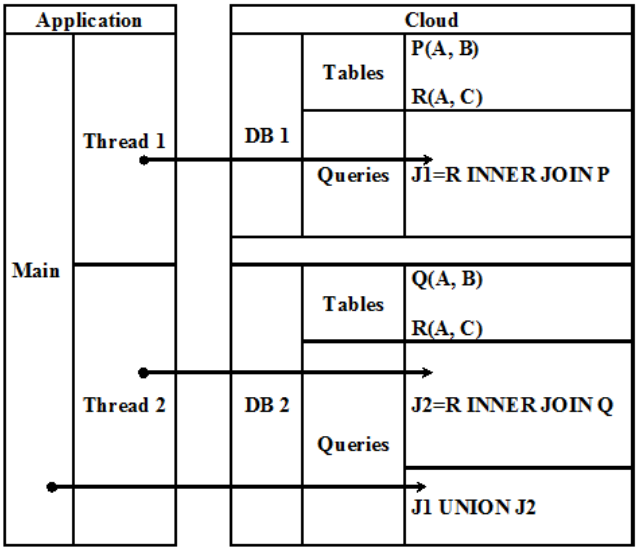


Рис. 5.9. Взаимодействие прикладной программы с БД в облаке

Прикладная программа может располагаться как на виртуальной машине в облаке, так и на компьютере пользователя. Во втором случае связь прикладной программы с базами данных осуществляется по глобальной сети. В обоих случаях связь прикладной программы и баз данных может быть реализована в модели ODBC. Первая база данных содержит таблицы R и P и запрос, в котором выполняется операция JOIN над этими таблицами. Вторая база данных содержит таблицы R и Q, запрос, в котором выполняется операция JOIN над этими таблицами и запрос, в котором выполняется операция UNION над результатами обоих JOIN-запросов. Прикладная программа запускает два параллельных потока, которые инициируют соответствующие им JOIN-запросы и отслеживают их состояния. Когда выполнение JOIN-запросов заканчивается, приложение запускает выполнение UNION-запроса. Базы данных обрабатываются независимо друг от друга, поэтому можно считать, что для решения задачи создан виртуальный программно-аппаратный комплекс. Его архитектура имеет много общего с такими распространенными архитектурами как симметричное мультипроцессирование (SMP) и массовый параллелизм (MPP).

Второй способ основан на параллельной реализации операций, которые составляют запрос, при использовании принципа симметричного горизонтального распределения (параграф 3.6).

Симметричное горизонтальное распределение таблиц P и Q между N базами данных фактически приводит к построению виртуального программно-аппаратного комплекса с MPP архитектурой, который располагается в облачной системе. Структура такого комплекса приведена на рисунке 5.10.

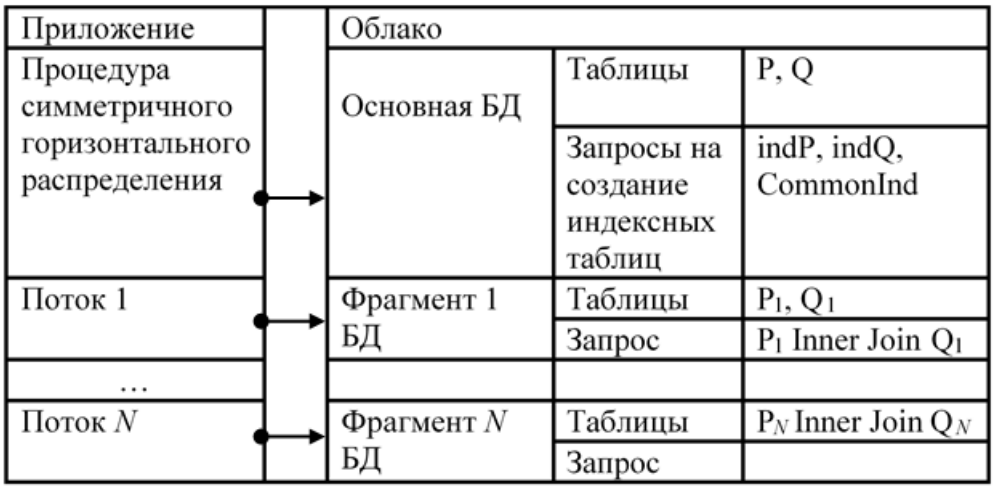


Рис. 5.10. Структура виртуального программно-аппаратного комплекса для массовой обработки данных

Для оценки качества распараллеливания операции JOIN был проведен вычислительный эксперимент. Виртуальный комплекс был построен на основе Windows Azure – платформы Microsoft, разработанной для реализации облачных вычислений. В состав комплекса входили пять независимых баз данных. Из них четыре использовались для симметричного горизонтального распределения таблиц. Результаты эксперимента приведены на рисунке 5.11.

Эксперимент проводился для таблиц P и Q, число строк в которых изменялось от 400000 до 2500000. Распределение таблиц производилось на основе алгоритма из раздела 3.5.3.1. На рисунке 5.11 приведены экспериментальные данные, подтверждающие тот факт, что применение предложенного автором метода параллельного выполнения операции JOIN дает существенное повышение производительности при реализации задач массовой обработки данных.

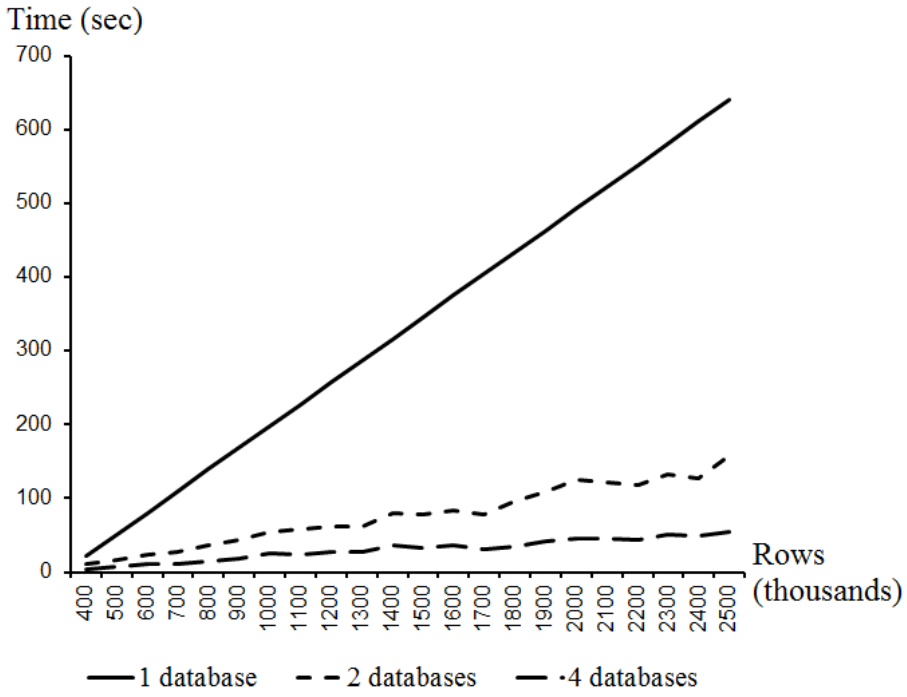


Рис. 5.11. Сравнительный анализ времени последовательного и параллельного выполнения операции JOIN

5.5. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием SMP-архитектуры в облачной среде

В этом эксперименте в облаке была создана виртуальная машина, на которой производилось сравнение реализации алгоритма черпака с фиксированным размером класса эквивалентности и реализации операции Join в Microsoft SQL Server. Для реализации предложенного параллельного алгоритма операции слияния нестрого упорядоченных файлов предложена архитектура программно-аппаратного комплекса, основанного на архитектуре SMP и реализованного в облачной системе Microsoft Azure на виртуальной машине с четырехядерным процессором Xeon E5-2673. В роли файлов выступали таблицы в базах данных СУБД Microsoft SQL Server.

В ходе эксперимента в основной базе данных последовательно создавались таблицы, соответствующие файлам X_K и Y_K . Таблицы содержали фиксированное число строк, но различные (увеличивающиеся на каждом этапе эксперимента) размеры классов эквивалентности. Эти базы разбивались на четыре фрагмента, каждый из которых содержал 25% таблиц основной базы данных.

Разбиение производилось на основе принципа симметричного горизонтального распределения.

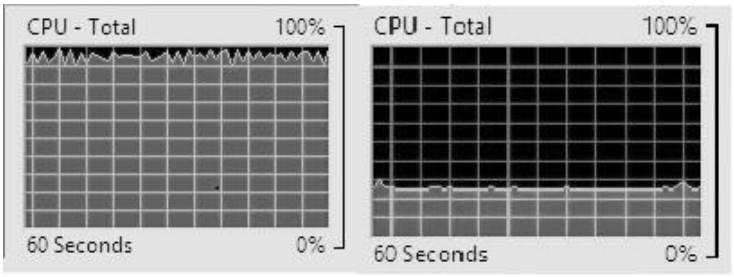
На каждом этапе эксперимента программа, реализующая алгоритм слияния нестрого упорядоченных файлов, четырьмя параллельными потоками считывала классы эквивалентности фрагментов таблиц. Каждый поток порождал два потока, которые выполняли декартово произведение этих классов эквивалентности. По завершении всех потоков, результаты объединялись в одну таблицу. Затем выполнялась операция JOIN над таблицами основной базы данных. При выполнении этой операции использовались стандартные средства распараллеливания, присущие СУБД Microsoft SQL Server.

Результаты эксперимента представлены на рисунках 5.11,5.12.

Рис. 5.12. Зависимость времени выполнения операции слияния нестрого упорядоченных файлов от размеров классов эквивалентности

График, приведенный на этом рисунке, показывает, что предложенные алгоритм и архитектура программно-аппаратного комплекса обеспечивают более эффективную обработку данных, чем стандартные средства СУБД.

На рисунке 5.12 показана загрузка процессоров при выполнении обоих способов реализации операции: А) – применение алгоритма черпака, Б) - стандартными средствами.



А) Б)

Рис. 5.12. График загрузки процессора

5.6. Анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием SMP-архитектуры и многоядерных графических процессоров

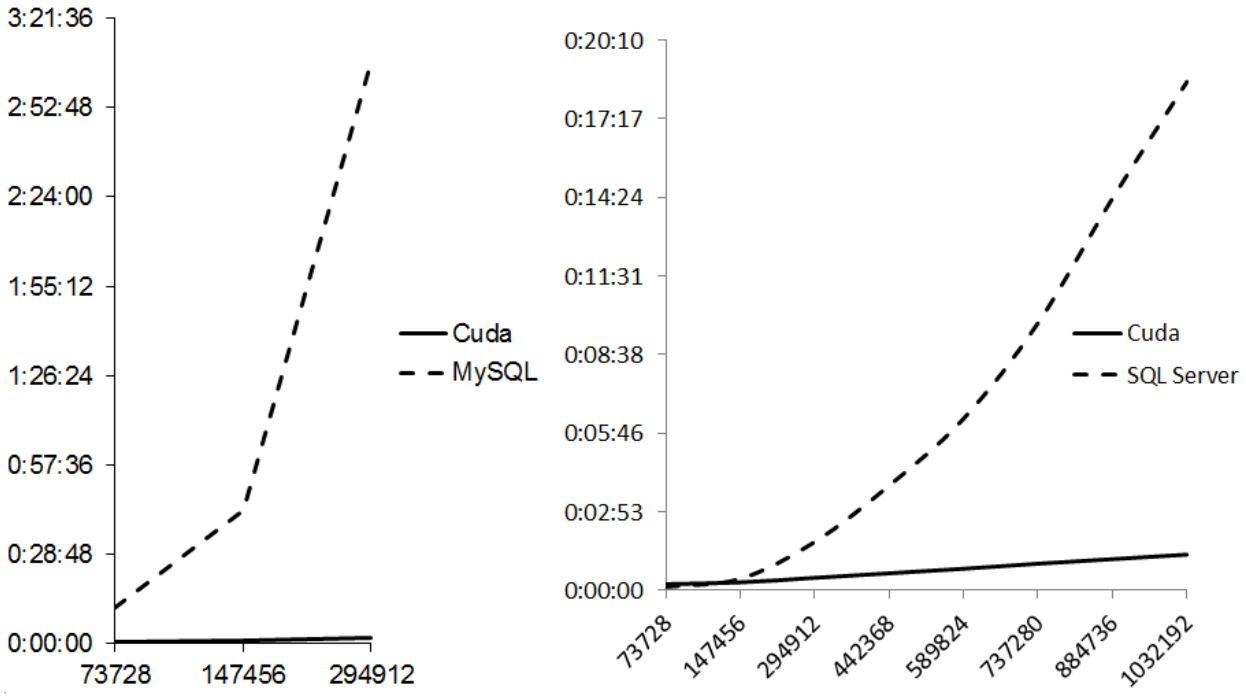
Здесь описан эксперимент с использованием двух вычислительных архитектур [203]. В первом случае операция JOIN была реализована над двумя таблицами с помощью технологии CUDA и принципа симметричного горизонтального распределения. Применение этого принципа позволило разработать алгоритм реализации операции JOIN, основанный на чтении (зачерпывании) исходных таблиц классами эквивалентности, соответствующими фиксированному значению экземпляра множества ключей. После зачерпывания оба класса эквивалентности передавались в видеопамять, и выполнялось их декартово произведение ядрами графического процессора. Во втором случае операция JOIN тех же таблиц осуществлялась средствами СУБД. Эксперимент поводился с использование двух СУБД: MySQL и Microsoft SQL Server 2014. Выбор этих СУБД определился тем, что они существенно различаются по производительности, кроме того СУБД Microsoft SQL Server 2014 многие действия реализует параллельно.

В ходе эксперимента была реализована операция, которой соответствует SQL-запрос:

SELECT T1.F1, SUM(T1.F2*T2.F2) FROM T1 INNER JOIN T2 ON T1.F1 = T2.F1 GROUP BY T1.F1 ORDER BY T1.F1.

Для эксперимента была использована 384-ядерная видеокарта Gigabyte GT730. Выполнялась операция JOIN над парами таблиц с одинаковым числом строк, которое изменялось от 73728 до 442368 (кратно числу ядер

графического процессора). Результаты, полученные в ходе эксперимента, приведены на рисунке 5.13 для СУБД MySQL и Microsoft Sql-Server.



А) Б)

Рис. 5.13. Сравнение времени выполнения операции JOIN средствами СУБД и технологии CUDA

5.7. Заключительные замечания к главе 5

В главе приведены результаты экспериментов, проведенных с целью проверки полученных в предыдущих главах результатов. А именно:

– анализ параллельной реализации операции умножения многомерных матриц;

– анализ параллельной реализации операции слияния нестрого упорядоченных файлов;

– анализ параллельной реализации операции слияния нестрого упорядоченных файлов средствами СУБД и SMP-архитектуры;

– анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием MPP-архитектуры в облачной среде;

– анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием SMP-архитектуры в облачной среде;

– анализ параллельной реализации операции слияния нестрого упорядоченных файлов с использованием SMP-архитектуры и многоядерных графических процессоров.

Проведенный экспериментальный анализ предложенных в предыдущих главах алгебраических методов массовой обработки данных показал их эффективность.

Основные результаты, полученные в данной главе, были опубликованы в работах [117, 145-147, 174-177, 188-192, 202, 203].

Глава 6. ПРИМЕНЕНИЕ АЛГЕБРАИЧЕСКОГО ПОДХОДА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

6.1. Использование предложенного метода для решения задач о кратчайшем пути

6.1.1 Решение традиционной задачи

Здесь рассматривается один из способов решения широкого класса оптимизационных задач, общее название которых "задачи о кратчайшем пути". Исходными данными для этих задач служат нагруженные графы. Нагруженный граф – это граф, в котором каждому ребру e присваивается число $w(e)$, называемое его весом. Задача о кратчайшем пути состоит в том, чтобы найти такой путь между двумя вершинами в графе, что сумма весов составляющих его ребер минимальна [204,205].

Как показано в параграфе 2.1.3, на практике к этому классу относятся различные задачи. Это могут быть как классические задачи о выборе пути с определенными свойствами (минимальной или максимальной длины), так и задачи, связанные с доступностью вершин, и задачи из области управления производством, например, определение входимости узлов и деталей в изделия (задача разузлования) [206]. Поэтому веса ребер графа могут быть элементами различных алгебраических систем. Выбор алгебраической системы определяет семантика предметной области, к которой принадлежит решаемая задача. Примеры алгебраических систем для различных задач приведены в таблице 6.1.

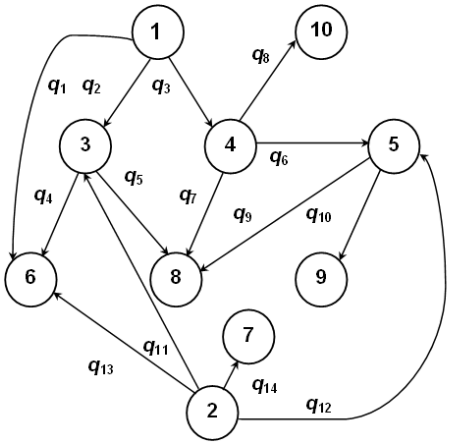
Таблица 6.1. Алгебраические системы для решения задач о выборе пути

Задача	Множество	Операции и нейтральные элементы	
		Аддитивная	Мультипликативная
Нахождение кратчайших путей в единицах длины	R^+	Min, ∞	+, 0
Нахождение путей с наибольшим числом пройденных вершин	{0, 1}	Max, 0	+, 0

Определение доступности вершин графа	{0, 1}	∨, 0	∧, 1
Разузлование проектируемого изделия	R_0	+, 0	×, 1

Для решения этих задач используются различные алгоритмы, например, алгоритм Флойда-Уоршалла. Он используется для нахождения кратчайших путей в нагруженном графе, а также для нахождения транзитивного замыкания некого отношения *R*. Выполнение алгоритма позволяет найти длины (суммарные веса) кратчайших путей между всеми парами вершин, хотя и не возвращает детали самих путей, например, в виде последовательностей вершин, через которые проходят эти пути.

Кроме этого алгоритма известны и другие, например, алгоритм, основанный на вычислении транзитивного замыкания матрицы весов графа *G*, которое вычисляется как $\hat{G} = \sum_{k=1}^{K} G^k, G^k \neq Z\ (i \le K), G^{K+1} = Z$ (Z – нуль-матрица) . Этот алгоритм основан на умножении матриц и поэтому легко распараллеливается. Известны эффективные алгоритмы параллельного умножения матриц, такие как алгоритм Фокса и алгоритм Кэннона. Но большинство достоинств этих алгоритмов теряются в том случае, когда матрица *G* разреженная, то есть содержит большое число нейтральных элементов. Пример такого графа и его матрицы весов показан на рисунке 6.1.



$$G = \begin{pmatrix} 0 & 0 & q_1 & q_2 & 0 & q_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_{11} & 0 & q_{12} & q_{13} & q_{14} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_4 & 0 & q_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_6 & 0 & 0 & q_7 & 0 & q_8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_9 & q_{10} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Рис. 6.1. Граф и разреженная матрица весов его ребер

Ранее были рассмотрены различные технологии хранения и обработки разреженных матриц [169]. Выбрана та из них, которая основана на том, что в памяти вычислительной системы матрица хранится как совокупность кортежей вида (***i***, ***j***, ***w***), ***i***, ***j*** – индексы элемента матрицы, ***w*** –вес ребра между вершинами ***i*** и ***j*** (значение ***w*** отлично от нейтрального элемента алгебраической системы, которой принадлежат веса ребер графа). Если использовать такое представление разреженных матриц, то их можно хранить и обрабатывать в реляционных базах данных. Далее рассмотрено использование реляционной модели SQL, для решения различных модификаций задачи о кратчайшем пути.

В параграфе 2.4.2 доказано соответствие алгебры многомерных матриц и реляционной алгебры. Более того, в рассматриваемом случае между этими алгебрами может быть установлено соответствие изоморфизма [110]. Действительно, каждой разреженной матрице соответствует единственное отношение со схемой $G(i, j, w)$. Все необходимые операции алгебры матриц могут быть реализованы операциями реляционной алгебры или их композициями. Операции умножения матриц соответствует композиция операций SELECT … GROUP BY… и JOIN. Запрос SELECT $G^k.i$, $G.j$, $Q(G.w \otimes G^k.w)$ AS w FROM G^k INNER JOIN G ON $G^k.j = G.i$ GROUP BY $G^k.i$, $G.j$; последовательно выполняет все возведения в степень таблицы G, получая в результате последовательность таблиц со схемой $G^k(i, j, w)$ (k = 1, …, K). Q – это функция, которая реализует групповую операцию, такую как Sum, Min, Max и им подобные. Таблица G^{K+1} не содержит ни одной строки и соответствует нуль-матрице Z.

Поскольку вычисление степеней матрицы весов G – циклический процесс, то для его реализации можно использовать язык программирования Transact-SQL. Фрагмент программы, вычисляющей кратчайшие пути в графе, приведен в листинге 6.1.

Листинг 6.1. Вычисление кратчайших путей в графе

```
1    INSERT INTO Gk SELECT * FROM G
2    SELECT @q = COUNT(*) FROM Gk
3    WHILE @q > 0
4    BEGIN
```

```
5         INSERT INTO @tcG SELECT * FROM Gk
6         INSERT INTO @GkNext SELECT Gk.i, G.j, Min(Gk.w + G.w) AS w
7           FROM Gk INNER JOIN G ON Gk.j = G.i
8           GROUP BY Gk.i, G.j ORDER BY Gk.i, G.j
9         DELETE * FROM Gk
10        INSERT INTO Gk SELECT * FROM @GkNext
11        SELECT @q = COUNT(*) FROM Gk
12        DELETE @GkNext
13    END
14    SELECT tcG.i, tcG.j, MIN(tcG.w) AS w FROM @tcG
15    GROUP BY tcG.i, tcG.j ORDER BY tcG.i, tcG.j
```

Здесь *q* – число строк в очередной вычисленной таблице. Таблицы и матрицы соответствуют друг другу следующим образом: G –матрица весов графа, Gk, GkNext – *k*-я и (*k*+1)-я степени матрицы весов, tcG – транзитивное замыкание матрицы весов.

Реализация алгоритма состоит из следующих шагов.

1. Таблице Gk присваивается первая степень матрицы G (строка 1);
2. Вычисляется число строк в таблице Gk (строка 2);
3. Выполняется основной цикл до тех пор, пока таблица Gk (очередная степень матрицы G) содержит хотя бы одну строку (строки 3-13);
4. В теле цикла:
   4.1. Строки таблицы Gk добавляются в таблицу tcG (транзитивное замыкание матрицы весов) (строка 5);
   4.2. Выполняется операция JOIN, которая реализует умножение матриц (строки 6-8);
   4.3. Таблице Gk присваивается очередная степень матрицы G, и вычисляется число строк в таблице Gk(строки 9-12);
5. Вычисляется окончательное значение транзитивного замыкания матрицы весов G (строки 14-15).

Итак, решение традиционных оптимизационных задач на графах, при условии, что матрица весов графа – разреженная матрица, возможно средствами баз данных. Однако в реальных условиях количество вершин в графе и соединяющих их ребер может быть настолько велико, что возникает проблема

обработки больших данных (Big Data). При этом в большинстве случаев время решения задач должно быть очень мало. Типичный пример такого рода реальных задач – это задача прокладки маршрута в навигационных системах, которая должна решаться каждый раз, когда изменяется ситуация, для большого числа движущихся объектов за очень короткий временной интервал.

На листинге 6.1 видно, что основная операция, используемая при решении задачи поиска кратчайших путей – это операция JOIN. Эта операция обладает большой вычислительной сложностью. Поэтому время ее реализации на больших данных велико. Поэтому целесообразно использование принципа симметричного горизонтального распределения. Его использование позволит существенно ускорить решение этой задачи для больших данных.

Из сказанного можно сделать вывод о том, что при решении оптимизационных задач, принадлежащих классу задач о кратчайшем пути возможно применение методов, основанных на базах данных. Можно сделать вывод о том, что эти методы наиболее эффективно применяются при использовании матричных алгоритмов в тех случаях, когда матрица весов графа разреженная. Также становится очевидным тот факт, что современные языки манипулирования данными позволяют эффективно реализовать матричные алгоритмы. Предложенный подход позволяет эффективно решать оптимизационные задачи на графах на основе использования принципа симметричного горизонтального распределения данных. Этот принцип дает возможность построения виртуальных программно-аппаратных комплексов, ориентированных на эффективное решение конкретных задач за счет параллельной реализации массовой обработки данных. Поскольку рассмотренный класс задач относится к этому типу обработки данных, повышение эффективности их решения становится возможным.

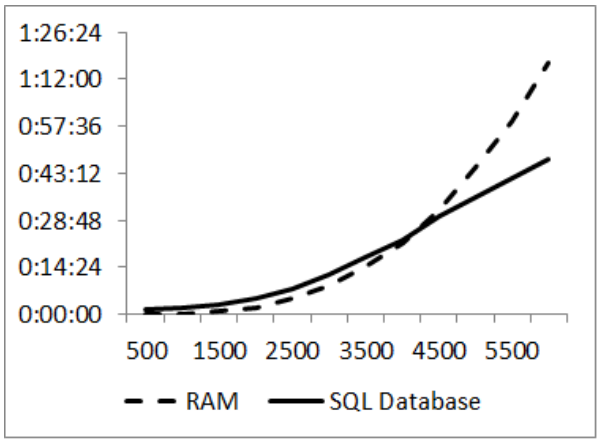


Рис. 6.2. Сравнение реализации алгоритма поиска кратчайших путей обычными средствами и средствами параллельной реализации в СУБД

На рисунке 6.2 приведены результаты эксперимента, проведенного для сравнения реализаций алгоритма поиска кратчайших путей обычными средствами и средствами параллельной реализации в СУБД. Эксперимент проводился для графов с большим числом вершин.

6.1.2. Решение задачи с одновременным построением пути

В этом разделе рассматривается применение предложенного метода к повышению эффективности обработки данных при решении задач построения маршрутов. Задачи построения маршрутов – неотъемлемая составная часть геоинформационных, логистических, и навигационных информационных систем. Кроме того, этот класс задач широко используется для построения и анализа различных коммуникационных сетей, например, телекоммуникационных и социальных. В частности, моделирование средствами граф-моделей телекоммуникационных сетей, состоящих из узлового сетевого оборудования и каналов связи, позволяет выбрать оптимальные маршруты передачи информации и схему соединения узлов коммутации, а также оценить пропускные способности линий [207, 208]. Моделирование социальных сетей позволяет эффективно оценить экономические и коммуникационные связи между пользователями, выполнить анализ процессов распространения информации, нахождения различных неформальных объединений и связанных подгрупп, на которые можно разбить общую сеть социальных взаимодействий [209, 210].

Современная особенность задач построения маршрутов заключается в необходимости обработки больших и сверхбольших данных (Big Data). Традиционный подход к их решению основан на математическом моделировании с использованием теории графов. При всей своей естественности для представления объектов предметных областей, их свойств и связей, этот подход обладает одним существенным недостатком. Как было показано в предыдущем параграфе, традиционные полиномиальные алгоритмы на графах позволяют находить свойства маршрутов: их существование между двумя вершинами, стоимости и тому подобные, но не позволяют получить сами маршруты как последовательности вершин графа в порядке их прохождения.

Большинство имеющихся подходов к решению задачи построения всех маршрутов в графе сводятся к переборным алгоритмам с экспоненциальной вычислительной сложностью. Для уменьшения сложности предлагаются различные способы, среди которых можно отметить решения на базе кластеризации [211] и применение генетических алгоритмов [212]. Причем последние ориентированы на случай графа, полностью помещающегося в оперативной памяти. Недостатки такого рода решений заключаются в том, что, во-первых, часто они основаны на эвристиках, применимых исключительно к условиям конкретной предметной области, что требует существенной переработки алгоритмов при переходе к другим предметным областям; во-вторых, для повышения их эффективности с использованием методов параллельного программирования приходится использовать сложные искусственные приемы.

Предложенный в работе подход позволяет:

– эффективное распараллеливание алгоритмов построения всех возможных маршрутов;

– использование технологии in database для построения всех маршрутов, что возможно в силу изоморфизма алгебры многомерных матриц и реляционной алгебры для рассматриваемого класса задач.

Решение задачи построения всех возможных маршрутов в ориентированном ациклическом графе легко реализуется на основе применения одного из

вариантов операции умножения многомерных матриц – $(1,0)$-свернутого произведения, при котором не производится суммирование элементов (отсутствуют кэлиевы индексы) [213-217]. Пусть матрица $G = \|w_{i_1 i_2}\|$ – матрица смежности (или весов ребер) графа. Элементы матрицы $w_{i_1 i_2}$ принадлежат некоторому типу, например, логическому или числовому и имеют значения отличные от нейтрального элемента этого типа, если в графе есть ребро $i_1 i_2$, и есть нейтральный элемент в противном случае. После умножения матрицы G. на себя получается трехмерная матрица $G^2 = \|w_{i_1 i_2 i_3}\|$. Здесь индекс i_1 обеих матриц-операндов – свободный (в матрице-результате он повторяется и второе его вхождение получает новое обозначение – i_3), а индекс i_2 – скоттов. Эта матрица содержит сведения обо всех маршрутах, проходящих по двум ребрам. После умножения матрицы G^2 на матрицу G получается четырехмерная матрица $G^3 = \|w_{i_1 i_2 i_3 i_4}\|$. В этой матрице индексы i_1, i_2 матрицы G^2 и индекс i_1 матрицы G (в матрице-результате он обозначается i_4) – свободные. В качестве скоттовых используются индекс i_3 матрицы G^2 и индекс i_2 матрицы G. Матрица G^3 содержит сведения обо всех маршрутах, проходящих по трем ребрам. Процесс продолжается до тех пор, пока не будет получена матрица, все элементы которой – нейтральные. Предшествующая ей матрица G^k содержит сведения обо всех маршрутах, состоящих из k ребер. Тогда, если элемент $w_{i_1^* i_2^* \ldots i_l^*}$ $(2 < l \leq k)$ отличен от нейтрального элемента, то последовательность значений индексов $i_1^*, i_2^*, \ldots, i_l^*$ есть последовательность вершин, через которые проходит маршрут, начинающийся в вершине i_1^*, заканчивающийся в вершине i_l^*, проходящий по l ребрам и обладающий свойством, определяемым значением $w_{i_1^* i_2^* \ldots i_l^*}$. Вычислительная сложность предложенного метода полиномиальная и имеет порядок $O(n^k + n^{k-1} + \ldots + n^3)$, где n – число вершин графа, k –число ребер в самом "длинном" маршруте.

Используя предложенный метод, можно строить маршруты с различными свойствами. К их числу относятся: поиск кратчайших по протяженности или стоимости маршрутов, например, поиск маршрутов, проходящих через мини-

мальное или максимальное число вершин, определение доступности вершин, производственные задачи построения спецификаций изделий.

Так же, как и в случае, рассмотренном в параграфе 6.1.1, для каждого конкретного случая специфическим будет только тип элементов матриц.

Пример 6.1. Пусть дан предложенный в параграфе 6.1.1 ориентированный ациклический граф (рисунок 6.1), и его матрица смежности *G* (таблица 6.2).

Таблица 6.2. Матрица *G* смежности графа

$i_2 \rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	1	1	0	0
5	0	0	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

Требуется для каждой пары вершин найти все связывающие их маршруты. Для решения поставленной задачи на этом 10-вершинном графе нужно построить реальную матричную машину EM=<M, {0, 1}; $+, \times, \vee, \wedge$; $M^k = Z$>. Здесь структура – это множество M многомерных матриц, индексы которых принимают значения от 1 до 10, для которых определены операции сложения и (1, 0)-свернутого произведения, а тип – это множество {0, 1} с операциями дизъюнкции и конъюнкции. Предикат $M^k = Z$ (Z – нуль-матрица) принимает значение ***истина*** если все элементы матрицы M^k равны 0 и значение ***ложь*** если хотя бы один ее элемент равен 1. После выполнения операции $^{1,0}(G \times G)$ получается трехмерная матрица G^2 , которая содержит все маршруты, проходящие по двум ребрам. В таблице 6.3 (и всех последующих таблицах) полностью приведены только содержащие единицы сечения этой и последующих матриц.

Таблица 6.3. Матрица G^2 всех маршрутов, проходящих по двум ребрам

											$\downarrow i_3$
0											1…4
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	5
1	0	1	0	1	0	0	0	0	0	0	
2…10	**0**										
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	6
1	0	0	1	0	0	0	0	0	0	0	
2…3	**0**										
4	0	0	0	0	0	0	0	1	0	0	
5…10	**0**										
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	7
1	0	0	0	1	0	0	0	0	0	0	
2…10	**0**										
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	8
1	0	0	0	1	0	0	0	0	0	0	
2…10	**0**										
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	9
1	**0**										
2	0	0	0	0	1	0	0	0	0	0	
3	0	0	0	0	0	1	0	0	0	0	
4	0	0	0	0	1	0	0	0	0	0	
5…7	**0**										
8	0	0	0	0	0	1	0	0	0	0	
9…10	**0**										
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	10
1…3	**0**										
4	0	0	0	0	0	0	0	1	0	0	
5…10	**0**										

Анализ полученной матрицы показывает, что из вершины 1 в вершину 5 ведут два маршрута (1, 2, 5) и (1, 4, 5), которым соответствуют равные единице элементы матрицы, w_{125} и w_{145}.

После выполнения операции $^{1,0}(G^2 \times G)$ получается четырехмерная матрица G^3 , которая содержит все маршруты, проходящие по трем ребрам.

Таблица 6.4. Матрица G^3 всех маршрутов, проходящих по трем ребрам

											$\downarrow i_3$	$\downarrow i_4$
0												1…5
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	8	6
1	0	0	0	1	0	0	0	0	0	0		
2…10	**0**											
0												7…8
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	5	9
1	0	1	0	1	0	0	0	0	0	0		
2…10	**0**											
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	6	
1	0	0	1	0	0	0	0	0	0	0		
2…3	**0**											
4	0	0	0	0	0	0	0	1	0	0		
5…10	**0**											
$i_2\rightarrow$ / $\downarrow i_1$	1	2	3	4	5	6	7	8	9	10	8	10
1	0	0	0	1	0	0	0	0	0	0		
2…10	**0**											

В этой матрице элементы w_{1259}, w_{1369}, w_{1459} имеют значение 1, что означает, что существует три маршрута, соединяющие вершины 1 и 9 и проходящие по трем ребрам.

После выполнения операции $^{1,0}(G^3 \times G)$ получается четырехмерная матрица G^4, все элементы которой, кроме w_{14869}=1, равны нулю. Последовательность значений индексов этого элемента соответствуют единственному маршруту, проходящему по четырем ребрам.

После следующего умножения выполняется предикат $^{1,0}(G^4 \times G) = G^5 = Z,$ что означает завершение процесса построения всех маршрутов.

Предложенный метод особенно эффективно работает в том случае, когда матрица смежности графа не разрежена, то есть содержит незначительное ко-

личество нейтральных элементов. В случае разреженных матриц эффективность существенно падает, так как вычислительная сложность остается без изменения. Выход из положения возможен за счет применения технологии in database. Тогда каждой многомерной матрице может быть поставлена в соответствие таблица базы данных по следующему правилу

$$M = \left\| w_{i_1 \ldots i_p} \right\| \leftrightarrow R(i_1 \ldots i_p, w).$$

Строки этой таблицы соответствуют только тем элементам матрицы, которые не равны нейтральному значению, и для каждого такого элемента содержат значения его индексов и его собственное значение.

Операции $(1, 0)$-свернутого произведения соответствует реляционная операция Join. В рассматриваемом случае умножению матрицы $M^k = \left\| w_{i_1 \ldots i_k} \right\|$ на матрицу $M = \left\| w_{i_1 i_2} \right\|$ соответствует запрос

SELECT $M_k.i_1$, …, $M_k.i_p$, $M.i_1$ AS i_{k+1}, $M_k.w$+M.w AS w

INTO M_{k+1}

FROM M_k INNER JOIN M ON $M_k.i_k$= M.i_1.

Следующий пример демонстрирует реализацию применения технологии in database для случая, рассмотренного в примере 1.

Пример 2. Матрице G ставится в соответствие таблица G и создается ее копия $G1$. Т таблица G и все последующие таблицы приведены в сводной таблице 4. После выполнения запроса

SELECT G1.i1, G1.i2, G.i2 AS i3, G1.w Or G.w AS w INTO G2

FROM G1 INNER JOIN G ON G1.i2 = G.i1

получается таблица $G2$, содержащая все маршруты, проходящие по двум ребрам.

Следующий запрос

SELECT G2.i1, G2.i2, G2.i3, G.i2 AS i4, G2.w Or G.w AS w INTO G3

FROM G2 INNER JOIN G ON G2.i3 = G.i1

создает таблицу $G3$, содержащую все маршруты, проходящие по трем ребрам.

Запрос

SELECT G3.i1, G3.i2, G3.i3, G3.i4, G.i2 AS i5, G3.w Or G.w] AS w INTO G4

FROM G3 INNER JOIN G ON G3.i4 = G.i1

создает таблицу *G*4, содержащую все маршруты, проходящие по четырем ребрам.

Таблица 6.5. Сводная таблица построения всех маршрутов в технологии in database

G			***G*$_2$**				***G*$_3$**					***G*$_4$**					
i_1	i_2	w	i_1	i_2	i_3	w	i_1	i_2	i_3	i_4	w	i_1	i_2	i_3	i_4	i_5	w
1	2	True	1	2	5	True	1	2	5	9	True	1	4	8	6	9	True
1	3	True	1	3	6	True	1	3	6	9	True						
1	4	True	1	4	5	True	1	4	5	9	True						
2	5	True	1	4	7	True	1	4	8	6	True						
3	6	True	1	4	8	True	1	4	8	10	True						
4	5	True	2	5	9	True	4	8	6	9	True						
4	7	True	3	6	9	True											
4	8	True	4	5	9	True											
5	9	True	4	8	6	True											
6	9	True	4	8	10	True											
8	6	True	8	6	9	True											
8	10	True															

Последим выполняется запрос

SELECT G4.i1, G4.i2, G4.i3, G4.i4, G4.i5, G.i2 AS i6, G4.w Or G.w AS w INTO G5

FROM G4 INNER JOIN G ON G4.i5 = G.i1;

Его результат – таблица $G_5(i_1, i_2, i_3, i_4, i_5, i_6, w)$, в которой должны быть отображены все маршруты, проходящие по пяти ребрам, не содержит ни одной строки. Это означает, что таких маршрутов нет, и процесс построения маршрутов завершен.

Кроме значительного сокращения объемов данных за счет избавления от разреженности, технология in database обладает важным достоинством, заключающемся в возможности распараллеливания обработки данных. Во-первых, во

всех современных системах управления базами данных имеются средства, обеспечивающие параллельную реализацию операции Join. Во-вторых, применение принципа симметричного горизонтального распределения данных позволяет существенно повысить эффективность распараллеливания за счет уменьшения объема данных в параллельно обрабатываемых фрагментах таблиц.

Из сказанного можно сделать следующие выводы:

1. Применение предложенного подхода существенно упрощает решение задач построения маршрутов, поскольку позволяет разрабатывать простые и удобные для понимания и программирования алгоритмы.

2. Простота алгоритмов, в свою очередь, обеспечивает возможность достаточно легкого распараллеливания на основе применения хорошо известных и проверенных методов.

3. Алгебраический подход обеспечивает эффективное сочетание технологии in memory и in database, что позволяет проектировать программно-аппаратные комплексы, наилучшим образом приспособленные для решения задач построения маршрутов.

6.2. Использование предложенного метода для решения задачи вывода ассоциативных правил

Здесь рассматриваются два метода повышения эффективности обработки данных при решении задач вывода ассоциативных правил. В отличие от большинства работ в этой области, в которых предлагаются методы улучшения запросов конечных пользователей, занимающихся анализом данных, здесь речь идет о методах, ориентированных на программиста разработчика аналитических информационных систем.

Ассоциативные правила в настоящее время превратились в мощный инструмент аналитических информационных систем. Поскольку в основе любых информационных систем лежат базы данных, весьма актуально направление, связанное с реализацией алгоритмов интеллектуального анализа на языках манипулирования данными. Поэтому реализации вывода ассоциативных правил

средствами СУБД посвящено достаточно много исследований, например, [218-220].

В общем виде задача вывода ассоциативных правил выглядит следующим образом. Имеется множество объектов $I=\{i_1, \ldots, i_n\}$ и множество свойств $P=\{p_1, \ldots, p_s\}$. Каждому объекту из I соответствует некоторое непустое подмножество свойств из P. Вывод ассоциативных правил состоит из двух этапов. На первом этапе для каждого подмножества свойств определяется количество объектов, которые обладают всеми свойствами этого подмножества, и только этими свойствами. На втором этапе, на основе полученных статистических данных и требований пользователя-аналитика, выводятся сами ассоциативные правила.

В работе рассмотрен только первый этап, который можно определить, как этап подготовки данных. Этот этап имеет высокую, как правило, экспоненциальную, вычислительную сложность. Повышение эффективности алгоритмов вывода ассоциативных правил достигается за счет распараллеливания известных алгоритмов, например, алгоритма Apriory [221]. Другой способ повышения эффективности алгоритмов состоит в использовании параллелизмов, реализованных в СУБД [222, 223]. Эти способы хороши для оптимизации запросов конечных пользователей, но они могут не учитывать всех возможностей вычислительных систем, на которых эти пользователи решают свои задачи. Для того, чтобы в конкретных организациях эти задачи решались эффективно, разрабатываются индивидуальные аналитические информационные системы. В этом случае программист-разработчик имеет возможность добиваться высокой эффективности за счет построения программно-аппаратного комплекса, ориентированного на конкретный класс задач.

Известны различные варианты реализации алгоритма Apriori средствами языка SQL [224- 227]. В дальнейшем рассматривается реализация, основанная на построении на каждой итерации промежуточной таблицы, которая используется на следующей итерации [228].

База данных для решения задачи вывода ассоциативных правил, как правило содержит следующий набор таблиц:

– $R_0(I, P_1)$ – исходные данные, каждая строка содержит идентификатор объекта и одно из его свойств;

– $CopyR_0(I, P_1)$ – копия таблицы R_0, она не обязательна и может использоваться для ускорения процесса обработки данных;

– $R_1(I, P_1, P_2)$, …, $R_{k-1}(I, P_1, …, P_k)$ – таблицы, получаемые на итерациях и содержащие данные для вывода ассоциативных правил на очередной итерации и используемые на следующей итерации как исходные данные.

Запрос, исполняемый на l-той итерации имеет, следующий общий вид:

Q_l = INSERT INTO R_l ($I, P_1, …, P_{l+1}$)

SELECT $R_{l-1}.I, R_{l-1}.P_1, …, R_{l-1}.P_l, R_0.P_1$ AS P_{l+1}

FROM R_{l-1} INNER JOIN R_0 [на итерации 1 – $CopyR_0$]

ON $R_{l-1}.I = R_0.I$ AND $\pi(R_{l-1}.I, R_{l-1}.P_1, …, R_{l-1}.P_l, R_0.P)$.

$\pi(R_{i\,l-1}.I, R_{l-1}.P_1, …, R_{l-1}.P_l, R_0.P)$ – предикат, запрещающий дублирование комбинаций свойств.

Таким образом, если для построения ассоциативных правил необходимо знать количества всех объектов, содержащих комбинации свойств от одного до $k \le s$, выполняются запросы Q_1 , …, Q_{k-1}. Из таблиц R_0, …, R_{k-1}, исходной и полученных в результате запросов, на втором этапе выводятся ассоциативные правила.

Ускорение подготовки данных возможно за счет того, что современные СУБД имеют возможность параллельного выполнения запросов, в том числе, и запросов, содержащих операцию JOIN. Предложенный подход обеспечивает возможность применения симметричного горизонтального распределения таблицы R_0 по ключу I между несколькими базами данных, расположенными на физически разных серверах. Это усилит эффект ускорения за счет существенного, практически, кратного числу серверов, уменьшения объемов фрагментов таблиц R_0, …, R_{k-1}. Кроме того, дополнительного эффекта можно достигнуть за

счет применения конвейерного метода для цепочки операций JOIN, которые реализуют вычисления на итерациях. Это потребует дополнительных усилий программиста-разработчика, но обеспечит существенное ускорение этапа подготовки данных, особенно в сочетании с симметричным горизонтальным распределением таблицы R_0.

Применение многомерно-матричной модели позволяет существенно улучшить временные характеристики этапа подготовки данных. Если номера объектов и свойств использовать в качестве индексов, то исходные данные в этой модели – матрица $M_0 = \left(m_{ij}^0\right), i = 1,...,n, j = 1,...,s.$ При условии, что $m_{ij}^0 = 1,$ если объект I_i обладает свойством P_j и 0 в противном случае, матрица M_0 взаимно однозначно соответствует таблице R_0. Операции JOIN в алгебре многомерных матриц соответствует операция (λ, μ)-свернутого произведения, которая позволяет умножать матрицы с произвольным числом измерений, получая многомерные матрицы. В рассматриваемом случае на первой итерации вычисляется трехмерная матрица $M_1 = {}^{1,0}(M_0 \times M_0) = \left(m_{ijj}^1\right), i = 1,...,n, j = 1,...,s,$ где $m_{ijj}^1 = m_{ij}^0 \times m_{ij}^0$ и двумерная матрица $T_1 = {}^{0,1}(M_0 \times M_0) = \left(t_{jj}^1\right), j = 1,...,s,$ где $t_{jj}^1 = \sum_{i=1}^{n} m_{ij}^0 \times m_{ij}^0.$ Матрица T_1 также может быть получена из матрицы M_1 операцией свертки, однако, обе матрицы можно вычислять одновременно. На l-той итерации вычисляются l+1-мерная матрица

$M_l = {}^{1,0}(M_{l-1} \times M_0) = \left(m^l_{i\underbrace{j...j}_{l}} \right), m^l_{i\underbrace{j...j}_{l}} = m^{l-1}_{i\underbrace{j...j}_{l-1}} \times m^0_{ij},$ которая будет использована на следующей итерации, и матрица $T_l = {}^{0,1}(M_{l-1} \times M_0) = \left(t^l_{\underbrace{j...j}_{l}} \right), t^l_{\underbrace{j...j}_{l}} = \sum_{i=1}^{n} m^{l-1}_{i\underbrace{j...j}_{l-1}} \times m^0_{ij},$ которая будет использована для вывода ассоциативных правил. Также, как и при использовании реляционной модели, для получения всех возможных комбинаций свойств от 1 до k необходимо выполнить k итераций. На последней итерации нет необходимости в построении матрицы M_k.

Для ускорения решения задачи вывода ассоциативных правил можно использовать предложенный параллельный алгоритм умножения многомерных матриц, который рассмотрен в параграфе 3.4.2. Кроме того, симметричное горизонтальное распределение в данном случае применимо для реализации рассматриваемой задачи в многомерно-матричной модели данных.

Для оценки предложенных методов был проведен вычислительный эксперимент. Анализу были подвергнуты три алгоритма вывода ассоциативных правил:

1. алгоритм Apriory, взятый из пакета "Apriori 1.1.1", который находится в центральном репозитарии модулей языка Python – Python Package Index [220];
2. алгоритм, реализованный средствами СУБД Microsoft Sql Server 2017 (на основе операции JOIN);
3. алгоритм на основе умножения многомерных матриц.

Программное обеспечение разрабатывалось в Microsoft Visual Studio 2017. Для алгоритма 1 исходные данные преобразовались из текстового файла в списковую структуру в оперативной памяти, необходимую для работы алгоритма. Для алгоритма 2 исходные данные располагались в таблице R_0, а подготовка к работе заключалась в удалении всех строк из промежуточных и результирующих таблиц. Для алгоритма 3 исходные данные преобразовывались из таблицы R_0 в матрицу M_0 в оперативной памяти. Программа, подготовки данных и вызова алгоритма 1 написана на языке Pyton. Алгоритм 2 реализован хранимыми процедурами, написанными на языке Transact Sql и вызываемыми из программы, написанной на языке C#. Алгоритм 3 реализован как процедура динамической библиотеки на языке C++, подготовка данных для которой и ее вызов осуществляются программой на языке C# [228].

Эксперимент производился на исходном наборе данных, содержащем миллион объектов, каждому из которых соответствовал набор от одного до пяти свойств. Таблица R_0 в этом случае содержала 2499677 строк. При обработке этих данных алгоритмы показали следующие результаты:

– Алгоритм 1 – 37 сек.

– Алгоритм 2 – 30 сек.

– Алгоритм 3 – 12 сек.

Для алгоритмов 2 и 3 было выполнено симметричное горизонтальное распределение данных между четырьмя серверами для алгоритма 2 и таким же количеством потоков для алгоритма 3. Число строк в фрагментах таблицы R_0 незначительно отличалось от 625000 в обе стороны. С учетом времени распределения и сбора результатов были получены следующие результаты;

– Алгоритм 2 – 14 сек.

– Алгоритм 3 – 4 сек.

При распределении вычислений по алгоритму 3 по восьми потокам общее время выполнения не превысило полутора секунд.

Из сказанного можно сделать следующие выводы:

1. В том случае, когда объемы данных настолько велики, что их невозможно разместить в оперативной памяти, целесообразно использовать алгоритмы подобные алгоритму 2.

2. Время решения задачи вывода ассоциативных правил таким способом сопоставимо с временем широко используемых в настоящее время алгоритмов типа алгоритма Apriory, а при использовании симметричного горизонтального распределения данных существенно меньше.

3. При возможности размещения данных в оперативной памяти алгоритмы, основанные на умножении многомерных матриц, значительно более эффективны и гораздо проще распараллеливаются, чем известные алгоритмы вывода ассоциативных правил.

4. Предложенные методы могут быть эффективно использованы при разработке программного обеспечения аналитических информационных систем.

6.3. Использование предложенного метода для решения задачи поиска изображений в базах данных

В этом параграфе рассматривается метод повышения эффективности решения задачи поиска изображений в базе данных. В настоящее время эта задача

актуальна для обработки больших данных во многих предметных областях, например, в геоинформационных системах [229-231].

Предложенный метод основан на использовании перцептивного хеширования изображений, трех уровней распараллеливания данных и процедур поиска изображений [232-234].

6.3.1. Архитектура программно-аппаратного комплекса для поиска изображений в базах данных

Подразумевается, что данные, необходимые для поиска изображений делятся на два типа: эталонные, долговременно хранящиеся в базе данные, среди которых осуществляется поиск, и оперативные данные – набор поисковых образов. Два изображения считаются близкими (сравнимыми, схожими), если расстояние Хэмминга между их ключами (значениями перцептивной хэш-функции от сравниваемых изображений) не превышает заданной величины.

Для уменьшения времени поиска предлагаются следующие подходы:

– сокращение перебора (access scan) за счет введения понятия ***вес ключа***;

– использование параллельной обработки данных программно-аппаратным комплексом с **SIMD** ***архитектурой*** на всех уровнях вычислений: от выборки данных из базы до сравнения идентификаторов.

Роль веса ключа в сокращении перебора состоит в том, что для вычисления расстояния Хэмминга используются не все ключи изображений, которые хранятся в базе. Ключ i-того поискового образа $\boldsymbol{w_i}$ сравнивается только с теми ключами изображений, хранящихся в базе, для которых выполняется условие $|\boldsymbol{w} - \boldsymbol{w_i}| \leq \boldsymbol{d}$, ($\boldsymbol{d}$ – фиксированное значение расстояния Хэмминга). Такой подход не исключает таких случаев, как, например, при $k = 3$ веса $\boldsymbol{w}_1 = 11110000$ и $\boldsymbol{w}_2 = 00001111$ равны, но расстояние Хэмминга между ними равно максимальному значению – 8. Однако, несмотря на это, эксперимент показал, что использование понятия веса ключа сокращает поиск и создает предпосылки для параллельной обработки данных.

Очевидно, что эталонных данных значительно больше, чем оперативных, поэтому предлагается способ распределения данных в программно-аппаратном комплексе, показанный на рисунке 6.3.

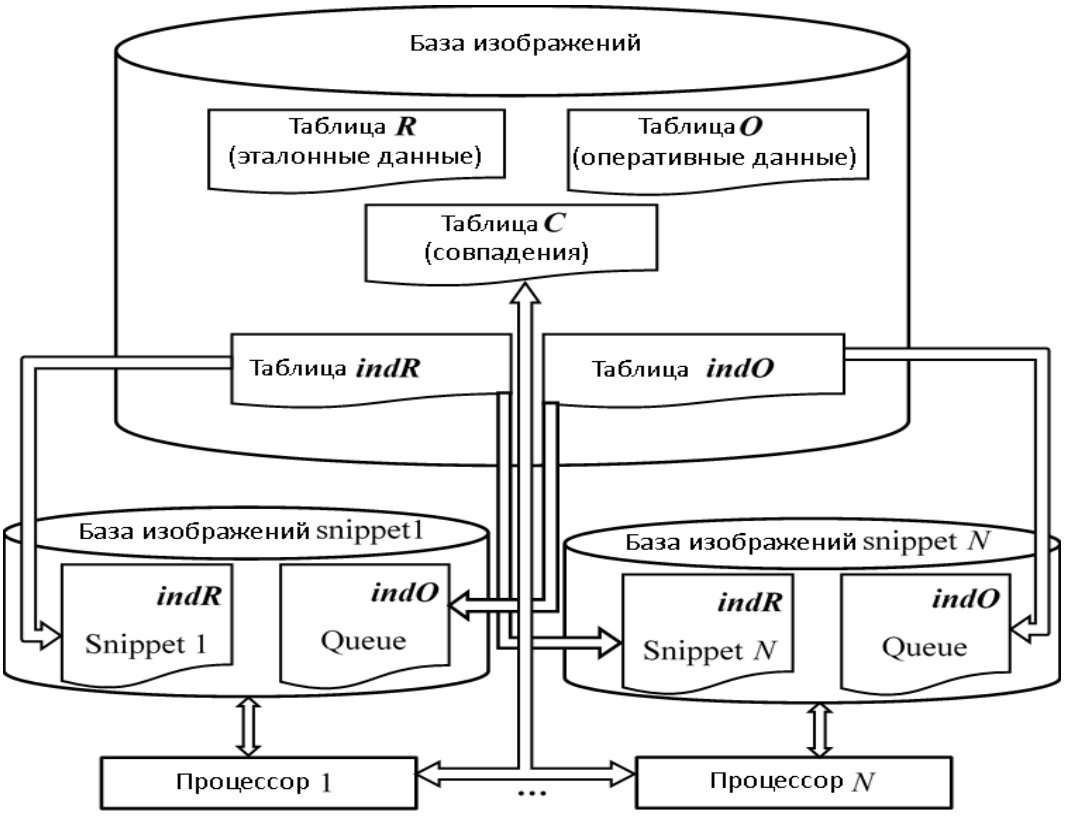


Рис. 6.3. Организация и распределение данных для поиска изображений

Независимо от того как организованы данные об изображениях в основной БД (предполагается, что они хранятся в таблицах: ***R*** – эталонные изображения и ***O*** – оперативные изображения), для решения задачи поиска изображений используются две соответствующие им таблицы метаданных – индексные таблицы ***indR***(K_{ph}, w) и ***indO***(K_{ph}, w). Они имеют одинаковые схемы, где:

– K_{ph} – ключ изображения (значение перцептивной хэш-функции от изображения):

– w – вес ключа, который вычисляется по формуле $w\ (K_{ph}) = \sum_{i=1}^{n} k_i$ (n – размерность ключа, k_i – значение координаты 0 или 1).

Каждому процессору ставится в соответствие фрагмент основной БД, который содержит фрагмент таблицы ***indR*** и фрагмент таблицы ***indO***. Фрагменты таблицы ***indR*** содержат, по возможности, одинаковое количество строк, кото-

рое определяется количеством процессоров. Следует заметить, что не обязательно выполнять создание и хранение полной таблиц ***indR*** в основной БД. Это требуется только в тех случаях, когда программно-аппаратный комплекс не имеет фиксированной структуры, например, при использовании грид-архитектуры, когда число процессоров может меняться от случая к случаю. В том случае, когда программно-аппаратный комплекс имеет архитектуру MPP с фиксированным числом процессоров, таблица ***indR*** может быть сформирована как совокупность ее фрагментов в фрагментах основной БД, размещенных во внешней памяти каждого процессора. Тогда переформирование таблицы ***indR*** необходимо только в случае изменения таблицы ***R***. Одновременно с таблицей ***indR*** создается таблица ***indSrv***, в которой содержится информация о распределении таблицы ***indR*** между процессорами. Это необходимо для последующего распределения таблицы ***indO***.

Создание таблицы ***indO*** выполняется при каждом решении задачи поиска изображений, поскольку оперативные данные меняются от случая к случаю. Здесь возможны два варианта ее организации. В первом, копии таблицы ***indO*** одновременно создаются в фрагментах основной БД на каждом процессоре. Во втором, в фрагментах основной БД создаются таблицы-очереди, которые пополняются из основной БД либо одновременно через фиксированные промежутки времени, либо по запросу процессора.

Распределение таблицы ***indR*** между процессорами осуществляется на основе принципа симметричного горизонтального распределения. В качестве ключа, по которому производится распределение, используется поле ***w*** этой таблицы.

6.3.2. Параллельное сравнение ключей изображений

Ключ изображения обычно представляет собой двоичные векторы с 2^k координатами. Обычно, k=6, 7, … . В памяти современных вычислительных они хранятся как 64-битовые слова или массивы таких слов. Пусть k_1 и k_2 – ключи изображений, HD(k_1, k_2) – расстояние Хэмминга между этими ключами. Срав-

нение изображений определяется соотношением HD(k_1, k_2)≤d (d ≥0). Для ускорения вычисления этого соотношения целесообразно использовать архитектурные особенности современных процессоров, которые состоят в возможности использования архитектуры SIMD на уровне регистров процессора. Эта возможность у процессоров с архитектурой x86 реализована в технологиях SSE и AVX. Эти технологии обеспечивают параллельное выполнение различных арифметических и логических операций над восьмью (шестнадцатью) 128-битовыми (256-битовыми) регистрами. Последний стандарт AVX-512 расширение до тридцати двух 512-битовых регистров. Для процессоров с архитектурой ARMv8 существуют SIMD-сопроцессоры, которые реализованы в технологии NEON. Эти сопроцессоры реализуют выполнение комбинированного 64-битовых и 128-битовых операций над тридцатью двумя 128-битовыми регистрами.

Далее приведен пример программы, которая реализует сравнение ключей с использованием регистров XMM. Эта программа сравнивает два 64-битных ключа поисковых изображений из оперативных данных с четырнадцатью ключами изображений, которые хранятся в базе данных. Она разработана на языке программирования C ++, а параллельная обработка реализована на языке ассемблера.

```
1 int i, j;
2 unsigned CmpRes[28];
3 unsigned *pB, *pS;
4 pB = BaseIm;
5 pS = SrchIm;
6 _asm
7 {
8       mov ESI, pB //Search Image pointer to [ESI]
9       movups  XMM0, [ESI] //Search Image  to xmm0
10      mov ESI, pS //Base Image pointer to [ESI]
11      mov ECX,0
```

```
12      movups  XMM1, [ESI][ECX] //Base Image  to XMM1
13      pxor    XMM1, XMM0
14      movups CmpRes[ECX], XMM1
15      add     ECX, 16
16                      …
17      movups  XMM7, [ESI][ECX] //Base Image  to XMM7
18      pxor    XMM7, XMM0
19      movups CmpRes[ECX], XMM7
20 }
21 for (i = 0; i < 14; i = i + 2)
22 {
23      HD7[i]=0;
24      HD7[i+1]=0;
25      for (j = 0; j < 2; j++) //Calc Hamming distance
26              while (CmpRes[2 * i + j] > 0)
27              {
28                      HD7[i]  = HD7[i]  + (CmpRes[2 * i + j] & 1);
29                      Res[2 * i + j] = Res[2 * i + j] >> 1;
30              }
31              while (Res[2 * (i + 1) + j] > 0)
32              {
33                      HD7[i]  = HD7[i]  + (Res[2 * (i + 1) + j] & 1);
34                      Res[2 * (i + 1) + j] = Res[2 * i + j] >> 1;
35              }
36 }
```

Программа начинается с объявления переменных, массива для сохранения результатов сравнения ключей и указателей, которым присваиваются значения параметров: адреса массивов, содержащих сравниваемые ключи. Оба ключа поисковых образов помещаются в регистр XMM0 (строки 8-9). Настраивается адрес массива ключей эталонных изображений и смещение относительно его

начала (строки 10-11). Затем семь раз повторяется последовательность действий, состоящая в присваивании очередному XMM регистру пары значений ключей эталонных изображений, вычисление несовпадающих координат операцией exjunction, сохранение результатов и увеличение смещения (строки 12-19). Остальные команды (строки 21-36) вычисляют расстояния Хэмминга и готовят возврат результатов вычислений в массиве HD14.

6.3.4. Реализация и анализ метода поиска изображений в базе данных

Для реализации и анализа предложенного метод поиска изображений в базе данных с использованием ключей изображений – значений перцептивной хэш-функции был проведен вычислительный эксперимент. Эксперимент проводился в следующих условиях:

– вычислительная система – рабочая станция с процессором Intel Core i7, оперативной памятью – 16 Гбайт, дисковой памятью – 1 Тбайт;

– программное обеспечение: операционная система Windows 10, система программирования Microsoft Visual Studio 2017, СУБД Microsoft Sql Server 2016;

– языки программирования Assembler, C++, C#, Transact SQL.

На первом этапе эксперимента была проведена оценка метода сравнения ключей с использованием регистров XMM. В оперативной памяти было случайным образом сгенерировано 14 миллионов. 64-битовых ключей. Для сравнения с ними одного ключа с использованием регистров XMM потребовалось 7 секунд, и 41 секунда без их использования. Из этого можно сделать вывод о том, что ускорение, практически, пропорционально числу используемых регистров XMM.

На втором этапе эксперимента исследовалась производительность предложенного метода распределенного поиска изображений в базе данных. Для этого была сделана оценка количества строк ($\boldsymbol{Q_i}$) в классах эквивалентности $\boldsymbol{indR_{w_i}}$ таблицы $\boldsymbol{indR}$. Очевидно, что верхняя граница

(супремум) $\boldsymbol{Q_i}$ определяется по формуле $C_n^{w_i} = \frac{n!}{w_i!(n-w_i)!}$ (n – размерность ключа). На рисунке 6.4 распределение значений $\boldsymbol{Q_i}$, для принятого в эксперименте 64-битового ключа показано штриховой линией. В реальных условиях $\boldsymbol{Q_i}$ принимает случайные значения, которые не больше супремума. На рисунке 6.4 случайное распределение значений $\boldsymbol{Q_i}$ показано сплошной линией. Количество классов эквивалентности в таблице ***indR*** определяется размерностью ключа. В эксперименте их шестьдесят три, так как каждый ключ содержит от одной до шестидесяти трех единиц.

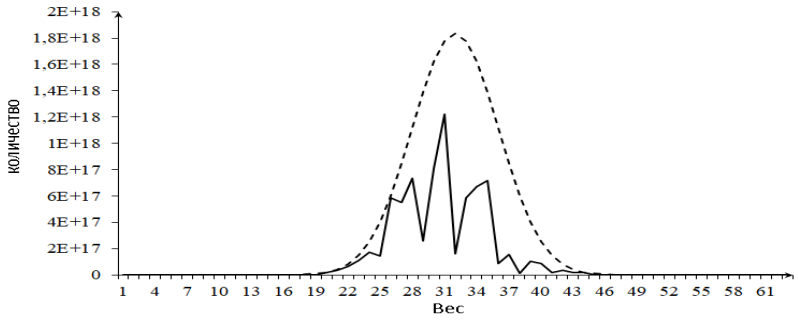


Рис. 6.4. Распределение значений $\boldsymbol{Q_i}$

Распределение классов эквивалентности, которое было использовано в эксперименте показано на рисунке 6.5. Для этого была создана таблица ***indR***, содержащая 3249491 случайным образом сгенерированных ключей.

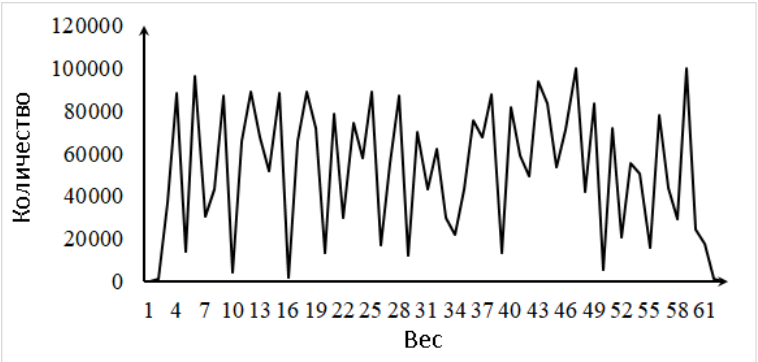


Рис. 6.5. Распределение значений $\boldsymbol{Q_i}$ для эксперимента

Симметричное горизонтальное распределение этой таблицы ***indR*** производилось по четырем фрагментам. Была использована хранимая процедура. Общее время выполнения процедуры, включая: удаление данных из таблиц-фрагментов, построение таблицы ***indRw*** и заполнение всех четырех

фрагментов таблицы ***indR*** данными составило 39 секунд. На рисунке 6.6 видно, что количество строк в таблицах-фрагментов примерно равны: ***Snp1R*** – 816830, ***Snp2R*** – 826194, ***Snp3R*** – 829139, ***Snp4R*** – 777328.

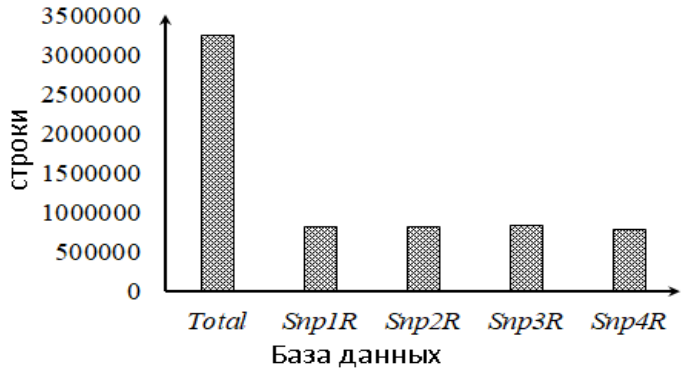


Рис. 6.6. Результат симметричного горизонтального распределения таблицы ***indR***

На третьем, последнем, этапе эксперимента было проведено исследование процедуры поиска изображений в базе данных. Таблица ***indO*** была сгенерирована пять раз с различным (возрастающим) числом строк: 12880, 17212, 25876, 51904, 129868. Эта таблица была распределена по четырем фрагментам. Затем, четырьмя потоками на четырех ядрах, была параллельно выполнена процедура слияния фрагментов таблиц ***indR*** и ***indO***. В ходе слияния, для всех пар ключей из таблиц ***indR*** и ***indO***, для которых выполняется условие $|w - w_i| \le d$, вычислялось расстояние Хэмминга. Если вычисленное значение не превышало ***d***, формировалась строка таблицы ***C***, содержащая оба ключа. Эти строки отправлялись в базу данных образов для дальнейшей обработки.

Рис. 6.7. Зависимость времени поиска от числа строк в таблице ***indO***

На рисунке 6.7 показана зависимость времени слияния таблиц ***indR*** и ***indO*** от числа строк в таблице ***indO***. На графике видно, что увеличение времени выполнения слияния пропорционально увеличению числа строк в ней. Из этого следует, что увеличение числа процессоров (горизонтальное масштабирование) существенно сокращает время поиска изображений.

Предложенный метод поиска изображений в базе данных и описание вычислительного эксперимента позволяют сделать следующие выводы.

– Использование перцептивного хеширования изображений позволяет выполнять операции сравнения не самих данных (изображений) а соответствующих им метаданных.

– Введение понятия "вес ключа" позволяет существенно сократить перебор ключей в процессе их сравнения.

– Использование принципа симметричного горизонтального распределения данных позволяет существенно сократить время сравнения ключей за счет уменьшения количества строк в фрагментах обрабатываемых данных и параллельной обработки этих фрагментов.

– Сокращение времени сравнения ключей также достигается за счет использования SIMD регистров процессоров и соответствующих им команд.

– Применение технологии "in database" позволяет использовать возможности параллельной обработки данных, реализованные в современных СУБД.

Сокращение времени поиска изображений пропорционально количеству процессоров, что свидетельствует о высокой масштабируемости программно-аппаратных комплексов, разрабатываемых на основе предложенной архитектуры.

6.4. Реализация алгоритма шифрования Хилла на основе алгебры многомерных матриц

Алгоритм шифрования Хилла [235] относится к числу популярных алгоритмов при симметричном шифровании, что объясняется его достаточно высокой криптостойкостью.

6.4.1. Краткое описание алгоритма шифрования Хилла

В общем виде этот алгоритм может быть представлен следующим образом. Пусть имеются алфавит A, символам которого соответствуют числовые коды, и два текста: T_1 – кодируемый текст, T_2 – кодирующий текст. Текстам (T_1, T_2) ставятся в соответствие матрицы (M_1, M_2), элементы которых – коды символов, составляющих эти тексты. Эти матрицы совместимы по умножению, и матрица M_2 имеет обратную матрицу M_2^{-1}. Кодирование текста T_1 состоит в вычислении матрицы $M_3 = M_1 \times M_2 \ mod \ mA$ (mA – количество символов в алфавите A). Декодирование состоит в вычислении матрицы $M_1 = M_3 \times M_2^{-1} \ mod \ mA$.

Анализ посвященных алгоритму шифрования Хилла публикаций показывает, что основные работы по его модификации и применению ведутся в двух направлениях.

1. Уменьшение времени кодирования/декодирования текста (ускорение).
2. Повышение устойчивости к различного рода атакам, таким как:

– атака с использованием только шифрованного текста, когда для расшифровки сообщение используется только шифрованный текст;

– атака с целью нахождения ключа при условии, что известны открытый и шифрованный тексты;

– атака с избранным открытым текстом, когда существует возможность отсылать любое количество простых текстов и получать в ответ соответствующие шифрованные тексты;

– атака с избранным шифрованным текстом, когда для подобранного открытого текста можно получить шифрованный текст, а для подобранного шифрованного текста – соответствующий открытый текст.

Решение проблем ускорения осуществляется за счет распараллеливания операций кодирования и декодирования [236]. Проблемы повышения крипто-

стойкости решаются посредством усложнения структуры кодирующей матрицы и операций над элементами матриц в операции их умножения [237-239].

Далее предложено использовать для кодирования больших текстов не обычные (плоские), а многомерные матрицы [240-242]. Выбор алгебры многомерных матриц может улучшить качество кодирования в силу следующих возможностей.

1. Увеличение криптостойкости за счет:

– особенностей операции умножения многомерных матриц, позволяющих задавать различные количества индексов, по которым производится сравнение и суммирование;

– существование для одной многомерной матрицы множеств различных детерминантов, единичных и обратных матриц (эти множества конечные).

2. Простое распараллеливание операций умножения многомерных матриц, вычисления их детерминантов и обратных матриц.

6.4.2. Дополнительные элементы алгебры многомерных матриц

Далее предполагается, что все элементы p-мерной матрицы $A = \left\| a_{i_1 \ldots i_p} \right\|$ – $A_{i_1 \ldots i_p} \in Z_0$ (точнее, его конечному подмножеству), что позволяет определить замкнутую на нем мультипликативную операцию и обратную к ней) и $i_1, \ldots, i_p = 1, \ldots, n$. Тогда, речь идет о целочисленной p-мерной матрице порядка n.

Элементы матрицы, взятые в количестве, не превосходящем ее порядка n, называются трансверсальными, если ни одна пара их не принадлежит одному и тому же простому сечению какой-либо ориентации. Совокупность n элементов матрицы, ни одна пара которых не принадлежит одному и тому же простому сечению какой-либо ориентации, образует ***трансверсаль***. Элементы трансверсали называются трансверсальными. Число всех трансверсалей матрицы A равно $(n!)^{p-1}$. Среди них находятся 2^p диагоналей, образованных элементами, расположенными на прямых, соединяющих противоположные вершины матрицы. Диагональ, у которой в каждом элементе значения всех индексов одинаковы,

называется главной, а ее первый элемент $a_{11\ldots1}$ – главным. На рисунке 6.8 приведен пример диагонали и трансверсали трехмерной матрицы порядка 4.

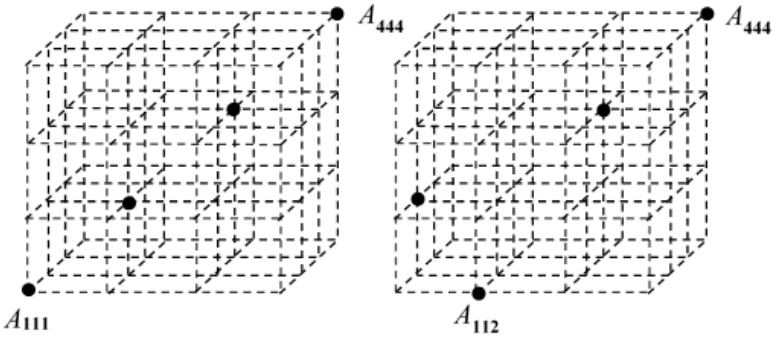


Рис. 6.8. Диагональ $A_{111},\ldots, A_{444}$ и транверсаль A_{112}, A_{211}, A_{333}, A_{444} трехмерной матрицы

Пусть:

– индексы $i_\nu^{(1)}, \ldots, i_\nu^{(n)}$, i_ν (ν = 1, 2, ..., p), образуют перестановку из чисел 1, 2, ..., n;

– $A_{i_1^{(1)},\ldots,i_p^{(1)}} \cdot \ldots \cdot A_{i_1^{(n)},\ldots,i_p^{(n)}}$ – произведение элементов некоторой трансверсали;

– $m \le p$ – четное неотрицательное число;

– I_μ – число инверсий в перестановке $i_\mu^{(1)},\ldots,i_\mu^{(n)}$ $(1 \le \mu \le m)$ значений индекса i_μ в произведении $A_{i_1^{(1)},\ldots,i_p^{(1)}} \cdot\ldots\cdot A_{i_1^{(n)},\ldots,i_p^{(n)}}$.

Индексы $i_{\mu_1}, \ldots, i_{\mu_m}$ называются альтернативными, остальные $p-m$ индексов – неальтернативными. Алгебраическая сумма $\sum_{j=1}^{n} (-1)^{\sum_{\mu=1}^{m} I_\mu} A_{i_1^{(1)},\ldots,i_p^{(1)}} \cdot\ldots\cdot A_{i_1^{(n)},\ldots,i_p^{(n)}}$ есть ***p-мерный детерминант матрицы A***. Поскольку число альтернативных индексов изменяется от 0 до p, число возможных детерминантов матрицы A определяется значением $\sum_{k=0}^{p} C_p^k$. Выражение $i_1^{\delta} \cdots i_p^{\delta}$ (δ=± для альтернативных индексов и δ=+ для неальтернативных) называется сигнатурой детерминанта.

Выражение $\left|{}^{\delta}_{i_1}\cdots{}^{\delta}_{i_p}\right| = \sum_{j=1}^{n}(-1)^{\sum_{\mu=1}^{m} I_\mu} A_{i_1^{(1)},\ldots,i_p^{(1)}} \cdot\ldots\cdot A_{i_1^{(n)},\ldots,i_p^{(n)}}$ определяет детерминант матрицы A с сигнатурой ${}^{\delta}_{i_1}\cdots{}^{\delta}_{i_p}$.

Пример 6.2. Четырехмерная матрица второго порядка $A = \left\|A_{i_1 i_2 i_3 i_4}\right\|$ имеет 16 детерминантов с разными сигнатурами. Среди них:

$$\left|A_{\substack{\pm\pm\pm\pm \\ i_1 i_2 i_3 i_4}}\right| = A_{1111}\cdot A_{2222} - A_{1112}\cdot A_{2221} - A_{1121}\cdot A_{2212} - A_{1122}\cdot A_{2211} - A_{1211}\cdot A_{2122} - A_{1212}\cdot A_{2121} -$$
$$-A_{122`}\cdot A_{2112} - A_{1222}\cdot A_{2111}\ (\textit{гипердетерминант});$$

$$\left|A_{\substack{++++ \\ i_1 i_2 i_3 i_4}}\right| = A_{1111}\cdot A_{2222} + A_{1112}\cdot A_{2221} + A_{1121}\cdot A_{2212} + A_{1122}\cdot A_{2211} + A_{1211}\cdot A_{2122} + A_{1212}\cdot A_{2121} +$$
$$+A_{122`}\cdot A_{2112} + A_{1222}\cdot A_{2111}\ (\textit{перманент});$$

$$\left|A_{\substack{+\pm+\pm \\ i_1 i_2 i_3 i_4}}\right| = A_{1111}\cdot A_{2222} + A_{1112}\cdot A_{2221} + A_{1121}\cdot A_{2212} - A_{1122}\cdot A_{2211} - A_{1211}\cdot A_{2122} + A_{1212}\cdot A_{2121} -$$
$$= -A_{122`}\cdot A_{2112} - A_{1222}\cdot A_{2111}\ (\textit{смешанный детерминант}).$$

Вычисление детерминанта многомерной матрицы рассмотрено в [235].

Поскольку в алгоритме шифрования Хилла используется только умножение матриц, то для повышения его криптостойкости имеет важное значение тот факт, что число всех возможных (λ, μ)-свернутых произведений p-мерной матрицы A на q-мерную матрицу B вычисляется по формуле $N_{p,q} = \sum_{\lambda+\mu=0}^{\min(p,q)} \frac{p!}{\lambda!\mu!(p-\lambda-\mu)!}\cdot\frac{q!}{\lambda!\mu!(q-\lambda-\mu)!}$.

Кроме того, для каждой многомерной матрицы можно определить некоторое количество единичных матриц, которые должны удовлетворять уравнениям вида ${}^{\lambda,\mu}(A\cdot E) = A$. Тогда количество индексов в разбиении m матрицы E должно равняться μ, а общее количество индексов в ней $\tau=\lambda + 2\mu$. Матрица $E = \left\|E_{csm}\right\|$ называется (λ, μ)-единичной матрицей матрицы A. $E_{csm} = \delta_{cm}$ (символ Кронекера) равняется 1, если все значения индексов разбиения c совпадают со значениями индексов разбиения m, и 0 в противном случае.

Количество (λ, μ)-единичных матриц для p-мерной матрицы равно $\frac{(p+1)\cdot(p+2)}{2}$.

Пример 6.3. Трехмерная матрица $A=\left\|A_{i_1i_2i_3}\right\|$ второго порядка имеет десять (λ, μ)-единичных матриц. Далее приводятся списки значений индексов единичных элементов некоторых из них ($\left\|E_{csm}\right\|,\left\|E_{s_1s_2cm}\right\|,\left\|E_{sc_1c_2m_1m_2}\right\|$).

Таблица1.Списки значений индексов единичных элементов единичных матриц

$\tau=\lambda+2\mu$	λ	μ	Индексы	Значения индексов
3	1	1	scm	111, 211, 122, 222
4	2	1	s_1s_2cm	1111, 1211, 1122, 1222, 2111, 2211, 2122, 2222
5	1	2	$sc_1c_2m_1m_2$	11111, 11212, 21111, 21212, 12121, 12222, 22121, 22222,

При определенных условиях p-мерная матрица $A=\left\|a_{i_1\ldots i_p}\right\|$ может иметь не менее одной обратной матрицы, поскольку ей может соответствовать несколько единичных матриц. При обычном разбиении индексов p-мерной матрицы $A=\left\|A_{lcs}\right\|$ порядка n и выбранной (λ', μ')-единичной матрице порядка n ($\tau'=\lambda'+2\mu'$, $0\le\lambda'+\mu'\le p$) (λ, μ)-обратная матрица (правая) $A^{-1}(\lambda,\mu)=\left\|A^{-1}_{csm}\right\|$ находится из уравнения $^{\lambda,\mu}(A\cdot A^{-1})=E(\lambda',\mu')$. Такая обратная матрица имеет $q=\tau'+\tau-p$ измерений (индексов). Из уравнения $^{\lambda,\mu}(A^{-1}\cdot A)=E(\lambda',\mu')$ находится левая обратная матрица. Для вычисления обратной матрицы многомерной матрицы достаточно, чтобы ее гипердетерминант был отличен от нуля. На его основе можно вычислять элементы обратной матрицы. Условия существования и алгоритмы вычисления (λ, μ)-обратной матрицы по заданной единичной матрице подробно изложены в [58]. Здесь же приводится простой пример ее вычисления.

Пример 6.4. Пусть $A=\|A_{lcs}\|$ трехмерная матрица второго порядка

$$A=\|A_{lsc}\|=\left\|\begin{array}{cc|cc} A_{111} & A_{112} & A_{121} & A_{122} \\ A_{211} & A_{212} & A_{221} & A_{222} \end{array}\right\|\begin{array}{l}\to(s)\\ \to(c)\\ \downarrow\\ (l)\end{array}$$

(λ=μ=1), а $E(3,0)=\left\|\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right\|\begin{array}{l}\to(s)\\ \to(c)\\ \downarrow\\ (l)\end{array}$ – (3, 0)-

единичная матрица (правая).

В этом случае $A^{-1}_{scm}=\left\|\begin{array}{cc|cc} \dfrac{A_{212}-A_{112}}{\left|\begin{array}{cc} A_{111} & A_{112} \\ A_{211} & A_{212}\end{array}\right|} & \dfrac{A_{212}-A_{112}}{\left|\begin{array}{cc} A_{111} & A_{112} \\ A_{211} & A_{212}\end{array}\right|} & \dfrac{A_{111}-A_{221}}{\left|\begin{array}{cc} A_{111} & A_{112} \\ A_{211} & A_{212}\end{array}\right|} & \dfrac{A_{111}-A_{221}}{\left|\begin{array}{cc} A_{111} & A_{112} \\ A_{211} & A_{212}\end{array}\right|} \\ \dfrac{A_{222}-A_{122}}{\left|\begin{array}{cc} A_{121} & A_{122} \\ A_{221} & A_{222}\end{array}\right|} & \dfrac{A_{222}-A_{122}}{\left|\begin{array}{cc} A_{121} & A_{122} \\ A_{221} & A_{222}\end{array}\right|} & \dfrac{A_{121}-A_{211}}{\left|\begin{array}{cc} A_{121} & A_{122} \\ A_{221} & A_{222}\end{array}\right|} & \dfrac{A_{121}-A_{211}}{\left|\begin{array}{cc} A_{121} & A_{122} \\ A_{221} & A_{222}\end{array}\right|} \end{array}\right\|\begin{array}{l}\to(s)\\ \downarrow\ \downarrow\\ (c)(m)\end{array}$. Эта

матрица существует, если определители $\left|\begin{array}{cc} A_{111} & A_{112} \\ A_{211} & A_{212}\end{array}\right|$ и $\left|\begin{array}{cc} A_{121} & A_{122} \\ A_{221} & A_{222}\end{array}\right|$ в знаменателях

отличны от нуля.

Кодирование в алгебре многомерных матриц производится следующим образом. Пусть A – матрица кодирующего текста, B – матрица кодируемого текста и E – единичная матрица для матриц A и B. Для создания конкретной системы кодирования необходимо определить следующие ее параметры:

– размерности (p, q) матриц A и B;

– место матрицы-операнда A в кодирующем произведении (левая – $A\cdot B$, правая – $B\cdot A$;

– значения κ, λ, μ, ν – количество индексов в разбиениях индексов матриц A и B;

– место обратной к A матрицы-операнда в произведении (${}^{\lambda,\mu}(A\cdot A^{-1})$ – правая или

${}^{\lambda,\mu}(A^{-1}\cdot A)$ – левая).

– λ', μ', необходимые для построения для построения обратной матрицы A^{-1}.

После определения параметров и создания матрицы B строятся матрицы: A, заполненная конкретными значениями кодирующего текста, соответствующая ей и матрице B (λ, μ)-единичная матрица и вычисляется обратная матрица A^{-1}.

В простейшем случае для возможности создания системы кодирования на основе алгебры многомерных матриц достаточно, чтобы матрицы A и B были одного порядка n и $\kappa=\lambda=\mu=\nu$. Тогда каждая из них содержит $n^{\kappa+\lambda+\mu} = n^{\lambda+\mu+\nu}$ элементов. В этом случае любая (λ, μ)-единичная матрица будет применима к обеим матрицам (умножение на единичную матрицу коммутативно). Для каждой из $\frac{(p+1)\cdot(p+2)}{2}$ (λ, μ)-единичных матриц существует $\frac{(p-2)\cdot(p-1)}{2}$ правых и столько же левых (λ', μ')-обратных матриц к матрице A. Из них выбирается единственная, которая будет использоваться для декодирования.

Из сказанного можно сделать два основных вывода.

1. Представление кодирующего и кодируемого текстов в виде многомерных матриц обеспечивает существенное ускорение основных процессов кодирования и декодирования, а именно:

– подготовка по заранее заданным значениям n, p, κ, λ, μ, и ν многомерной матрицы, обратной к кодирующей матрице;

– умножение матрицы кодируемого текста на кодирующую матрицу и матрицы закодированного текста на матрицу, обратную к кодирующей.

Ускорение происходит за счет параллелизма алгоритмов всех операций над многомерными матрицами. Причем эти алгоритмы обладают высокой степенью масштабирования.

2. Предложенный метод существенно повышает криптоскойкость кодирования. Этот вывод основан на том, что число вариантов параметров многомерных матриц кодирующего и кодируемого текстов n, p, κ, λ, μ, ν, количество детерминантов, единичных и обратных матриц задаются факториальными

функциями. Например, при большом количестве 2^{24} символов в кодирующем тексте и n=2 и p=24 возможное число (λ, μ)-свернутых произведений равно 1360135024740956026161. Следовательно, алгоритмы, реализующие все виды атак, будут иметь вычислительную сложность порядка $O(N!)$, где значение N определяется значениями n и p. Даже при незначительном возрастании значений этих параметров многомерной матрицы вычислительная сложность их подбора, вычисления необходимых детерминантов, подбора единичной и вычисления обратной матрицы многократно возрастает.

6.5. Заключительные замечания к главе 6

В главе рассмотрено применение предложенного подхода к решению практических прикладных и системных задач массовой обработки данных.

Первый раздел этой главы посвящен использования предложенного метода для решения задач о кратчайшем пути. Рассмотрено решение традиционной задачи матричной формы метода Флойда-Уоршелла. Показано, что при последовательном возведении матрицы весов ребер графа в (1, 0)-свернутую степень, набор значений индексов отличного от нейтрального элемента k-той степени этой матрицы есть последовательность номеров вершин графа. Это есть путь, начинающийся в вершине, номер которой соответствует значению первого индекса, заканчивающийся в вершине, номер которой соответствует значению последнего индекса, и проходящий через k–2 вершины. Значение элемента есть стоимость этого пути. Рассмотрен случай, когда число ребер таково, что матрица весов сильно разрежена. На основе того факта, что для этой задачи алгебра многомерных матриц и реляционная алгебра изоморфны, предложено ее решение средствами языка Transact-SQL. Вычислительная сложность этих алгоритмов достаточно высока, однако использование предложенных методов распараллеливания умножения многомерных матриц и операции Join, позволяет получить результаты за приемлемое время.

Во втором разделе показано, что использование предложенного в первом разделе метода возведения матрицы весов графа в (1, 0)-свернутую сте-

пень может быть использовано для решения задачи вывода ассоциативных правил.

В третьем разделе рассмотрено использование предложенного метода для решения задачи поиска изображений в базах данных на основе перцептивного хеширования. Каждому изображению ставится в соответствие двоичное число 64 и более двоичных цифр (хеш-код), которое служит идентификатором (ключом) изображения в БД. Два изображения считаются близкими, если расстояние Хэмминга между их хэш-кодами не превосходит заданного значения. Введено понятия веса ключа, которое позволило использовать метод симметричного горизонтального распределения для параллельной обработки запросов к БД. Предложен и реализован метод параллельного сравнения ключей изображений с использованием SIMD-регистров процессора. Предложена архитектура программно-аппаратного комплекса для поиска изображений в базах данных.

Четвертый раздел посвящен реализации алгоритма шифрования Хилла на основе алгебры многомерных матриц. Приведены дополнительные элементы алгебры многомерных матриц: понятие трансверсали, детерминанта, обратной многомерной матрицы. Показано, что криптостойкость обобщенного алгоритма Хилла повышается за счет свойств многомерных матриц, таких как количество и размерности индексов, выбора скоттовых и кэлиевых индексов, возможности построения различных единичных и, соответственно, обратных матриц.

Основные результаты, полученные в данной главе, были опубликованы в работах [117, 213-217, 228, 232-234, 240-242].

Заключение

В диссертационной работе были рассмотрены вопросы построения программно-аппаратных комплексов для реализации массовой обработки данных на основе алгебраических моделей и методов. Предложен новый метод формализации моделей данных и моделей вычислений, основанный на универсальных многоосновных алгебраических системах и объектно-ориентированном подходе к проектированию и разработке программно-аппаратных комплексов для решения задач массовой обработки данных. Предложена и разработана теоретико-множественная (файловая) алгебраическая система, которая используется для доказательства соответствия известных и используемых на практике моделей данных и моделей вычислений. Описана алгебра многомерных матриц и предложен метод – абстрактная алгебраическая машина, который позволяет использовать в качестве элементов многомерных матриц различные, как простые, так и структурные (кортежи), типы данных. Разработаны алгебраический и аксиоматический методы доказательства соответствия моделей данных и моделей вычислений. Проведено доказательство гомоморфизма, а для конкретных задач – изоморфизма, теоретико-множественной, многомерно-матричной и реляционной моделей на основе использования обоих методов. Проведено исследование проблемы повышения эффективности процессов МОД и разработано обобщение алгоритма выбора последовательности операций умножения матриц методом динамического программирования для (λ, μ)-свернутого произведения многомерных матриц. Проведено доказательство того, что синтез оптимизированного процесса МОД может быть реализован методом динамического программирования. Приведен пример синтеза такого процесса. Рассмотрена проблема повышения эффективности процессов МОД на основе параллельной реализации операций. Предложена стратегия повышения эффективности процессов МОД на основе выбора модели данных и модели вычислений в зависимости от степени разреженности данных. Разработаны этапы построения программно-аппаратных комплексов, для реализации МОД. Разработана архитектура программно-аппаратного комплекса для реализации многомерно-

матричной модели данных. Разработаны архитектуры программно-аппаратных комплексов для реализации простых (однопроходных) операций теоретико-множественной модели данных. Разработана архитектура программно-аппаратного комплекса для параллельной реализации операции слияния нестрого упорядоченных файлов в теоретико-множественной и реляционной моделях данных для различных алгоритмов, в том числе с использованием ассоциативных вычислительных систем. Проведен экспериментальный анализ предложенных архитектур, подтвердивший их эффективность. Показана эффективность предложенных методов МОД для решения различных прикладных задач.

Публикации по теме диссертации

Основные результаты диссертации полностью опубликованы в работах [52, 57, 62, 66, 67, 78, 79, 110, 121-123, 138, 139, 143-147, 151, 155-157, 167-170, 181-187, 195, 196, 206-210, 221, 225-227, 233-235].

Направления дальнейших исследований

Теоретические исследования и практические разработки, выполненные в рамках диссертационной работы, предполагается продолжить по следующим направлениям.

1. Разработка системы автоматизации проектирования и программирования процессов массовой обработки на основе многомерно-матричной и реляционной моделей с применением предложенной системы оптимизации.
2. Разработка многомерно-матричной машины баз данных с использованием современных облачных средств создания виртуальных кластеров на основе тензорных и графических процессоров и современных программных средств, реализующих алгебру тензоров.
3. Разработка на основе тех же аппаратных средств абстрактных (универсальных) алгебраических машин для реализации обработки различных структур данных с произвольными элементами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Опарин Г. А., Новопашин А. П. Булевы модели синтеза параллельных планов решения вычислительных задач //Вестник Новосибирского государственного университета. Серия: Информационные технологии. – 2008. – Т. 6. – №. 1.

2. Ледянкин И. А., Легков К. Е. О некоторых концептуальных вопросах разработки параллельных структур вычислительных задач кластерных вычислительных систем //Наукоемкие технологии в космических исследованиях Земли. – 2014. – Т. 6. – №. 6.

3. de Carvalho Silva J., de Oliveira Dantas A. B., de Carvalho Junior F. H. A Scientific Workflow Management System for orchestration of parallel components in a cloud of large-scale parallel processing services //Science of Computer Programming. – 2019. – Т. 173. – С. 95-127.

4. Ordonez C., Bellatreche L. A Survey on Parallel Database Systems from a Storage Perspective: Rows Versus Columns //International Conference on Database and Expert Systems Applications. – Springer, Cham, 2018. – С. 5-20.

5. Liu C. et al. Ghostrider: A hardware-software system for memory trace oblivious computation //ACM SIGPLAN Notices. – 2015. – Т. 50. – №. 4. – С. 87-101.

6. Рабинович З. Л. Машинный интеллект и ЭВМ пятого поколения. – Киев. – Кибернетика. – №3. – 1985. – С. 95-107.

7. Рабинович З. Л. Развитие архитектур ЭВМ в связи с их интеллектуализацией //Разработка ЭВМ нового поколения: архитектура, программирование, интеллектуализация. – 1986. – С. 18-26.

8. Sharifani K., Amini M. Machine learning and deep learning: A review of methods and applications //World Information Technology and Engineering Journal. – 2023. – Т. 10. – №. 07. – С. 3897-3904.

9. Prince S. J. D. Understanding deep learning. – MIT press, 2023.

10. Soori M., Arezoo B., Dastres R. Artificial intelligence, machine learning and deep learning in advanced robotics, a review //Cognitive Robotics. – 2023. – Т. 3. – С. 54-70.

11. Tandon A. et al. Retraining Convolutional Neural Networks for Specialized

Cardiovascular Imaging Tasks: Lessons from Tetralogy of Fallot //Pediatric cardiology. – 2021. – Т. 42. – №. 3. – С. 578-589.

12. Pietron M., Wielgosz M. Retrain or Not Retrain? – Efficient Pruning Methods of Deep CNN Networks //International Conference on Computational Science. – Springer, Cham, 2020. – С. 452-463.

13. Калиниченко Л. А., Рывкин В. М. Машины баз данных и знаний. – Наука. Гл. ред. физ.-мат. лит., 1990.

14. DeWitt D., Gray J. Parallel database systems: The future of high performance database systems //Communications of the ACM. – 1992. – Т. 35. – №. 6. – С. 85-98.

15. Fey M. et al. Relational deep learning: Graph representation learning on relational databases //arXiv preprint arXiv:2312.04615. – 2023.

16. Robinson J. et al. RelBench: A Benchmark for Deep Learning on Relational Databases //arXiv preprint arXiv:2407.20060. – 2024.

17. Zhou L. et al. Serving Deep Learning Models from Relational Databases //Advances in Database Technology-EDBT. – 2024. – Т. 27. – №. 3. – С. 717-724.

18. Zahradník L., Neumann J., Šír G. A deep learning blueprint for relational databases //NeurIPS 2023 Second Table Representation Learning Workshop. – 2023.

19. Özsu M. T., Valduriez P. Distributed and parallel database systems //ACM Computing Surveys (CSUR). – 1996. – Т. 28. – №. 1. – С. 125-128.

20. Лапаев А. О. О параллельном вычислении дискретного преобразования Фурье и проведённых экспериментах //Вестник российских университетов. Математика. – 2010. – Т. 15. – №. 1.

21. Chicheva M. A. Parallel computation of multidimensional discrete orthogonal transforms reducible to a discrete Fourier transform //Pattern Recognition and Image Analysis. – 2011. – Т. 21. – №. 3. – С. 381-383.

22. Григорьев Ю. А., Плужников В. Л. Оценка времени соединения таблиц в параллельной системе баз данных //Информатика и системы управления. – 2011. – №. 1. – С. 3-16.

23. Chu S., Balazinska M., Suciu D. From theory to practice: Efficient join query evaluation in a parallel database system //Proceedings of the 2015 ACM SIGMOD

International Conference on Management of Data. – 2015. – С. 63-78.

24. Voevodin V. V. et al. Supercomputer Lomonosov-2: Large scale, deep monitoring and fine analytics for the user community //Supercomputing Frontiers and Innovations. – 2019. – Т. 6. – №. 2. – С. 4-11.

25. Антонов А. С. и др. Исследование динамических характеристик потока задач суперкомпьютерной системы //Вычислительные методы и программирование. – 2013. – Т. 14. – С. 104-108.

26. Ибрагимов Т. Р., Мунерман В. И. Возможность использования процессоров ARMV8 для параллельных вычислений //Системы компьютерной математики и их приложения. – 2018. – №. 19. – С. 152-157.

27. You X. et al. Performance evaluation and analysis of linear algebra kernels in the prototype tianhe-3 cluster //Asian Conference on Supercomputing Frontiers. – Springer, Cham, 2019. – С. 86-105.

28. Зверев М. М. Технологии высокопроизводительных вычислений с использованием графического ускорителя //Образование и наука в России и за рубежом. – 2019. – №. 9. – С. 76-79.

29. Галимов М. Р., Биряльцев Е. В. Некоторые технологические аспекты применения высокопроизводительных вычислений на графических процессорах в прикладных программных системах //Вычислительные методы и программирование. – 2010. – Т. 11. – №. 3. – С. 77-93.

30. Кондрашев В. А., Волович К. И. Управление сервисами цифровой платформы на примере услуги высокопроизводительных вычислений //Математическое моделирование и информационные технологии в инженерных и бизнес-приложениях. – 2018. – С. 217-223.

31. Dziekonski A. et al. Implementation of matrix-type FDTD algorithm on a graphics accelerator //MIKON 2008-17th International Conference on Microwaves, Radar and Wireless Communications. – IEEE, 2008. – С. 1-4.

32. Yong K. K., Karuppiah E. K., See S. C. W. Galactica: a GPU parallelized database accelerator //Proceedings of the 2014 International Conference on Big Data Science and Computing. – 2014. – С. 1-4.

33. Jouppi N. et al. TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings //Proceedings of the 50th Annual International Symposium on Computer Architecture. – 2023. – С. 1-14.

34. Басс А. В., Антонов М. А. Работа с ПЛИС с использованием языка описания аппаратуры Verilog //Известия Тульского государственного университета. Технические науки. – 2019. – №. 3.

35. Солдатов А., Костеж А. Расширяемая вычислительная платформа Peleновая архитектура компании Xilinx //Компоненты и Технологии. – 2010. – №. 113 . – С. 12-14.

36. Шалагин С. В. Реализация параллельной сортировки массива чисел методом Хоара в архитектуре ПЛИС/FPGA //Системы компьютерной математики и их приложения. – 2020. – №. 21. – С. 237-242.

37. Давиденко А. Н., Гильгурт С. Я. Алгоритмы распознавания строк в системах обнаружения вторжений на ПЛИС //Моделювання та інформаційні технології. – 2010.

38. Shu R. et al. Direct Universal Access: Making Data Center Resources Available to {FPGA} //16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19). – 2019. – С. 127-140.

39. Gupta P. K. Accelerating datacenter workloads //26th International Conference on Field Programmable Logic and Applications (FPL). – 2016. – Т. 2017. – №. 9. – С. 20.

40. Thomas J., Lavin C., Kaviani A. Software-like Compilation for Data Center FPGA Accelerators //Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Tech-nologies. – 2021. – С. 1-6.

41. Каляев И. А. и др. Реконфигурируемые вычислительные системы на основе ПЛИС семейства Virtex-6 //Вестник Уфимского государственного авиационного технического университета. – 2011. – Т. 15. – №. 5 (45).

42. Singh M., Leonhardi B. Introduction to the IBM Netezza warehouse appliance //Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research. – 2011. – С. 385-386.

43. Stolze K., Beier F., Dimov V., Kalogeiton E., Tošić M. IBM Data Gate: Making On-Premises Mainframe Databases Available to Cloud Applications // Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 2023. – p. 119-230.

44. Леонтьев А. В., Шершаков А. В., Янакова Е. С. Российское программно-аппаратное решение распознавания эмоционального состояния людей для интеллектуальных экосистем //Наноиндустрия. – 2020. – №. S96-1. – С. 125-128.

45. Беляев А. А., Янакова Е. С., Тюрин А. А., Мачарадзе Г. Т. Анализ видеоинформации с использованием векторных потоковых процессоров с общей памятью //Известия Тульского государственного университета. Технические науки. – 2020. – №. 10. – С. 254-263.

46. Фролова С. Е., Янакова Е. С. Методы достижения максимальной эффективности платформы прототипирования высокопроизводительных систем на кристалле на задачах искусственного интеллекта //Наноиндустрия. – 2020. – №. S96-2. – С. 585-588.

47. Янакова Е. С., Смирнов Д. П. Методика имитационного объектно-ориентированного моделирования автоматизированных производственных процессов на базе модифицированных E-сетей //Оборонный комплекс-научно-техническому прогрессу России. – 2016. – №. 1. – С. 15-21.

48. Янакова Е. С., Мачарадзе Г. Т., Гагарина Л. Г., Швачко А. А. Параллельно-конвейерная обработка видеоинформации в многопроцессорных гетерогенных системах на кристалле //Известия высших учебных заведений. Электроника. – 2021. – Т. 26. – №. 2. – С. 172-183.

49. Ecker W., Müller W., Dömer R. Hardware-dependent software //Hardware-dependent Software. – Springer, Dordrecht, 2009. – С. 1-13.

50. Arató P., Mann Z. A., Orbán A. Algorithmic aspects of hardware/software partitioning //ACM Transac-tions on Design Automation of Electronic Systems (TODAES). – 2005. – Т. 10. – №. 1. – С. 136-156.

51. Takahashi D. Fast Fourier transform algorithms for parallel computers. – Springer Singapore, 2019. DOI //doi.org/10.1007/978-981-13-9965-7.

52. Воеводин В.В. Вычислительная математика и структура алгоритмов. –

М.: Изд-во МГУ, 2006. – 112 с. – ISBN 5-211-05310-9.

53. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. – Киев: Наукова думка, 1989. – 376 с. – ISBN 5-12-000499-7.

54. Giloi W. K. Programming Models and Tools for Massively Parallel Computers //Massively Parallel Processing Applications and Development. – Elsevier, 1994. – С. 3-14.

55. Danelutto M., Mencagli G., Torquati M., González–Vélez H., Kilpatrick, P. Algorithmic Skeletons and Parallel Design Patterns in Mainstream Parallel Programming //International Journal of Parallel Programming. – 2021. – Т. 49. – №. 2. – С. 177-198.

56. Falgout R. D., Jones J. E. Multigrid on massively parallel architectures //Multigrid Methods VI. – Springer, Berlin, Heidelberg, 2000. – С. 101-107.

57. Weigel M. Monte Carlo methods for massively parallel computers //Order, Disorder and Criticality: Advanced Problems of Phase Transition Theory. – 2018. – С. 271-340.

58. Соколов Н.П. Введение в теорию многомерных матриц. – Киев: Наукова думка, 1972 г. – 176 с.

59. Goncharov E., Iljin P., Munerman V. Multidimensional Matrix Algebra Versus Tensor Algebra or μ> 0 //2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2020. – С. 1949-1954.

60. Гончаров Е.И., Мунерман В.И., Синицын И.Н. Современные технологические средства создания многомерно-матричных машин баз данных. – «Системы высокой доступности» Т. 17 №1 за 2024 г. 5-17.

61. Lombardi L. Mathematical structure of nonarithmatic data processing procedures. – Journal of ACM, 1962, v. 9, n.1. – p. 136-159. Русский перевод: Ломбарди Л. Математическая структура процедур обработки нечисловой информации/ Современное программирование. Сборник статей. – М. Радио и связь, 1967. – 5-39.

62. Обработка информационных массивов в автоматизированных системах управления / В. М. Глушков, В. П. Гладун, Л. С. Лозинский, С. Б. Погребинский

; под общ. ред. акад. В. М. Глушкова ; АН УССР, Ин-т кибернетики. – К. : Наукова думка, 1970. – 181 с. : табл., рис.

63. Codd E.F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, v. 13, n. 6, 1970. – p. 377-387.

64. Митенков В. Б., Митенков К. А., Мунерман В. И. Параллельная подготовка данных для расчета виброзащитных характеристик в ходе летных испытаний //Системы высокой доступности. – 2018. – Т. 14. – №. 5. – С. 46-49.

65. Munar A., Chiner E., Sales I. A big data financial information management architecture for global banking //2014 international conference on future internet of things and cloud. – IEEE, 2014. – С. 385-388.

66. Sun N. et al. iCARE: A framework for big data-based banking customer analytics //IBM Journal of Research and Development. – 2014. – Т. 58. – №. 5/6. – С. 4:1-4:9.

67. Meer, Kamran H. Best Practices in ERP Software Applications. – Lincoln, NE: iUniverse, 2005. – 232 c. – ISBN 0-595-31513-1

68. O‘Leary D. L. Enterprise resource planning systems. – Cambridge University Press, 2000. – 232 c. – ISBN 0-521-79152-9.

69. Синицын И. Н., Шаламов А. С. Лекции по теории интегрированной логистической поддержки. – 2-е изд., перераб., и доп. – М.: ТОРУС ПРЕСС. – 2019. – 1072 с.: ил. ISBN 978-5—94588-267-6.

70. Когаловский М. Р. Энциклопедия технологий баз данных. М.: Финансы и статистика, 2002. – 800 с. – ISBN 5-279-02276-4.

71. Гарсиа-Молина Г., Ульман Д., Уидом Д. Системы баз данных. Полный курс.: Пер. с англ.: – М.: Издательский дом ""Вильямс", 2004, - 1088 с., с ил. – ISBN 5-8459-0384-X;

72. Кузнецов С.Д. Основы баз данных. - М.: Изд-во "Интернет-университет информационных технологий – ИНТУИТ.ру", 2005. – 488 с.: ил. – http://citforum.ru/database/osbd/contents.shtml.

73. Гендель Е. Г., Мунерман В. И. Применение алгебраических моделей для синтеза процессов обработки файлов. – Управляющие системы и машины, Ки-

ев: Наукова думка. 1984. № 4. С.69-72.

74. Гендель Е. Г., Мунерман В. И., Шкляр Б.Ш. Оптимизация процессов обработки данных на базе алгебраических моделей. – Управляющие системы и машины, Киев: Наукова думка, 1985. № 6. с.91-95.

75. Ульман Д. Базы данных на Паскале: Пер. с англ. – Машиностроение, 1990.

76. Мунерман В. И., Мунерман, Д. В., Синицын, И. Н., Чукляев И. И., Параллельная реализация задач интегрированной логистической поддержки (CALS) //Современные информационные технологии и ИТ-образование. – 2014. – №. 10. – С. 548-554.

77. Нуриев М. Г., Ахмадуллин Т. Р. Организация параллельной обработки SQL-запросов с использованием системы управления базами данных MS SQL Server //Актуальные проблемы науки и образования в условиях современных вызовов (шифр–МКАП 24). – 2023. – С. 69-77.

78. Karwaczyński P., Sitko M., Pietras S., Marczuk B., Wasielewski M., Kwiatkowski J., Fraś M. Impact of Design Decisions on Performance of Embarrassingly Parallel. NET Database Application //Vietnam Journal of Computer Science. – 2024. – С. 1-22.

79. Бёрнс Б. Распределенные системы. Паттерны проектирования. – Питер, 2022. 224 с. – ISBN:978-5-4461-0950-0.

80. Vega F. F., Cantú-Paz E. (ed.). Parallel and Distributed Computational Intelligence. – Springer, 2010. – Т. 269.

81. Srinivasan V., Gooding A., Sayyaparaju S., Lopatic T., Porter K., Shinde A., Narendran B. Techniques and Efficiencies from Building a Real-Time DBMS //Proceedings of the VLDB Endowment. – 2023. – Т. 16. – №. 12. – С. 3676-3688.

82. Flynn MJ. Very High Speed Computing Systems // Proc. IEEE. – 1966. – Vol. 54. – P. 1901-1909.

83. Flynn M.J. Computer Organization and Architecture // Operating Systems, An Advanced Course. – Springer, 1978 (Lecture Notes in Computer Science; Vol.60). – P. 17-98.

84. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 536 с.

85. Левин Н. А., Мунерман В. И. Метод логически последовательного доступа к данным //Системы высокой доступности. – 2011. – Т. 7. – №. 4. – С. 65-67.

86. Комиссарова А. Н., Мунерман В. И. Мера вычислительной сложности массовой обработки данных //Системы высокой доступности. – 2011. – Т. 7. – №. 4. – С. 68-71.

87. Flynn M.J. Rudd K. W. Parallel architectures // ACM Computing Surveys. 1996. – Vol. 28, No. 1. – P. 67-70.

88. Компьютеры, вычисления и параллелизм: [Электронный ресурс] // Вычислительный центр им. А.А. Дородницына РАН Федерального исследовательского центра «Информатика и управление» РАН. URL: http://www.ccas.ru/paral/contents.html/ (Дата обращения: 26.03.2023).

89. Netezza Performance Server SQL command reference [Электронный ресурс] IBM. URL: https://www.ibm.com/docs/en/netezza?topic=dud-netezza-performance-server-sql-command-reference/ (Дата обращения: 26.03.2023).

90. A Platform for High Performance Data Warehousing and Analytics: [Электронный ресурс] IBM. URL: http://www.redbooks.ibm.com/redpapers/pdfs/redp4725.pdf/ (Дата обращения: 26.03.2023). Русский перевод: http://www.redbooks.ibm.com/redbooks/pdfs/redp4725-00-ru.pdf.

91. An introduction to data mesh [Электронный ресурс] IBM. URL: http://www.ibm.com/developerworks/data/library/techarticle/0203lurie/0203lurie.html/ (Дата обращения: 26.03.2023).

92. A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server [Электронный ресурс] IBM. URL: http://www.oracle.com/technetwork/database/exadata/exadata-technical-whitepaper-134575.pdf/ (Дата обращения: 26.03.2023).

93. Андреев А.Н., Воеводин В.В., Жуматий С. А. Кластеры и суперкомпьютеры – близнецы или братья. – М.: Открытые системы № 05-06, 2000.

94. Foster I., Kesselman C., Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations International Journal of High Performance Computing Applications Volume 15 Issue 3, August 2001, Pages 200 – 222.

95. Демичев А.П., Ильин В.А., Крюков А.П. Введение в грид-технологии. – Препринт НИИЯФ МГУ - 2007 - 11/832. – Москва 2007.

96. Лымарь Т.Ю., Соколинский Л.Б. Инкапсуляция параллелизма в исполнителе запросов СУБД Омега // Высокопроизводительные вычисления и их приложения: Труды Всероссийск. науч. конф. (30 октября - 2 ноября 2000 г., г. – М.: Изд-во МГУ, 2000. – С. 136-140.

97. Лымарь Т.Ю., Соколинский Л.Б. Организация параллельного исполнителя запросов на базе многопроцессорного вычислительного комплекса МВС-100/1000 // Вестник Челябинского университета. Сер. 3. Математика, механика, информатика. – 2002. – №1(6). – С. 177-188.

98. Лымарь Т.Ю., Соколинский Л.Б. Управление потоками данных в параллельном исполнителе запросов СУБД Омега для МВС-100/1000 //Распределенные комплексы: оптимизация и приложения в экономике и науках об окружающей среде (DSO'2000). Сб. докл. к Междунар. конф. (Екатеринбург, 30 мая - 2 июня 2000 г.). – Екатеринбург: УрО РАН, 2000. – С. 326-328.

99. Бурцев В.С. Вычислительные процессы с массовым параллелизмом. – Журнал «Электроника: НТБ», № 2, 2002 г. – с. 32-35.

100. Ширай А.Е., Провоторова А.О., Гайдаенко Т.И. Аппаратная поддержка векторно-матричной модели данных//Системы компьютерной математики и их приложения: материалы XIII международной научной конференции, посвященной 75-летию профессора Э.И. Зверовича. – Смоленск: Изд-во СмолГУ, 2012. – Вып. 13. – 256 с. ISBN 978-5-88018-445-3, продолжающееся издание. – с. 135-138.

101. Гантмахер Ф. Р. Теория матриц. – М.: "Наука", 1966 г. – 576 с.

102. Fox G.C., Otto S.W., Hey A.J.G Matrix Algorithms on a Hypercube I: Matrix Multiplication. – Parallel Computing. 1987, 4 H. – p. 17-31.

103. Hall P.A.V. Optimization of a Single Relational Expression in a Relational Da-

ta Base System IBM J. R&D. – May, 1976. – 20, № 3.

104. Chaudhuri S. An overview of query optimization in relational systems //Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. – 1998. – C. 34-43.

105. Bellamkonda S., Ahmed R., Witkowski A., Zait M., Lin C. Enhanced Subquery Optimizations in Oracle // Proceedings of the 35th international conference on Very large data base, 2009, pp. 1366-1377. Перевод Леонида Борчука Под ред. Сергея Кузнецова URL: http://citforum.ru/database/oracle/subquery_opt/.

106. Мацей Пилеки (Maciej Pilecki) Оптимизация производительности запросов SQL Server [Электронный ресурс] IBM. URL: http://technet.microsoft.com/ru-ru/magazine/2007.11.sqlquery.aspx/ (Дата обращения: 26.03.2023).

107. Оптимизация запросов в используемых приложениях с помощью структур планов SQL Server 2008 R2 [Электронный ресурс] IBM. URL: http://technet.microsoft.com/ru-ru/library/ms187032(v=sql.105).aspx/ (Дата обращения: 26.03.2023).

108. Improve performance for your data virtualization data sources with remote connectors. Make queries across data sources present in remote data centers [Электронный ресурс] IBM. URL: http://www.ibm.com/developerworks/ru/library/dm-0703kapoor/ (Дата обращения: 26.03.2023).

109. Смирнов А. В. Проблема оптимизации запросов в расширяемой СУБД SciDB //Электронный журнал Алгоритмы, методы и системы обработки данных. – 2011. – № 3. URL: http://amisod.ru/index.php?option=com_content&view=article&id=69:amisod-2011-3-18-smirnov-2&catid=15:amisod-3-18-2011&Itemid=111.

110. Oracle для профессионалов. Архитектура, методики программирования и особенности версий 9i, 10g и 11g. Пер. с англ./ТомКайт. – М.: "Вильямс" 2011. – 848 стр. ISBN 978-5-8459-1703-4, 978-1-43-022946-9;

111. Хранимые процедуры (компонент Database Engine) [Электронный ресурс] Microsoft. URL: http://technet.microsoft.com/ru-ru/library/ms190782.aspx/ (Дата

обращения: 26.03.2023).

112. Багуи С. Объектно-ориентированные базы данных: достижения и проблемы. – «Открытые системы», № 03, 2004.

113. Liskov B., Zilles S. Programming with abstract data types. – ACM SIGPLAN Notices Volume 9 Issue 4, April 1974. – P. 50 – 59.

114. Дейт К., Дарвен Х. Основы будущих систем баз данных. Третий манифест. М: Янус-К, 2004. – 656с. – ISBN: 5-8037-0183-1.

115. Курош А.Г. Лекции по общей алгебре. – М.: Наука, 1973 г. – 400 с.

116. Мальцев А. И. Алгебраические системы. – Издательство" Наука", Главная редакция физико-математической литературы, 1970.

117. Мунерман, В. И. Массовая обработка данных. Алгебраические модели и методы: монография / В.И. Мунерман. — Москва: ИНФРА-М, 2023. — 229 с. — (Научная мысль). — DOI 10.12737/1906037. - ISBN 978-5-16-018035-9. - Текст : электронный. - URL: https://znanium.com/catalog/product/1906037 (дата обращения: 06.11.2023). – Режим доступа: по подписке.

118. Грэхем И. Объектно-ориентированные методы. Принципы и практика/ Object-Oriented Methods: Principles & Practice. – 3-е изд. – М.: «Вильямс», 2004. – С. 880. – ISBN 0-201-61913-X.

119. Бадд Т. Объектно-ориентированное программирование в действии – An Introduction to Object-Oriented Programming. – СПб.: «Питер», 1997. – 464 с. – ISBN 5-88782-270-8

120. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ – Object-Oriented Analysis and Design with Applications / Пер. И.Романовский, Ф.Андреев. – 2-е изд. – М., СПб.: «Бином», «Невский диалект», 1998. – С. 276–278. – 560 с. – ISBN 5-7989-0067-3.

121. Cattell R. G. G. et al. (ed.). The object data standard: ODMG 3.0. – Morgan Kaufmann, 2000. ISBN 1-55860-647-5.

122. Barry D. K., Duhl J. Object Storage Fact Books: Object DBMSs and Object-Relational Mapping. [Электронный ресурс], Barry & Associates, Inc. URL: https://www.barryandassociates.com/reports/object-relational.html/ (Дата обраще-

ния: 26.03.2023).

123. DataTable Класс [Электронный ресурс] Microsoft. URL: https://learn.microsoft.com/ru-ru/dotnet/api/system.data.datatable?view=net-8.0/ (Дата обращения: 26.03.2023).

124. Агафонов В.Н. Спецификации программ: понятийные средства и их организация. – Новосибирск: Наука, 1987 г. – 240 с.

125. Емельченков Е. П., Мунерман, В. И., Мунерман, Д. В., Самойлова, Т. А. Объектно-ориентированный подход к разработке моделей данных //Современные информационные технологии и ИТ-образование. – 2020. – Т. 16. – №. 3. – С. 564-574.

126. Ахо А. В., Хопкрофт Дж. Э., Ульман Дж. Д. Структуры данных и алгоритмы. – М.: Вильямс, 2010 г. – 382 с. – ISBN 978-5-8459-1610-5, 0-201-00023-7.

127. Важоньи А. Научное программирование в промышленности и торговле / Важоньи Андрю; Пер.с англ. В.В.Головинского, Е.М.Четыркина; Вступ.ст. А.Я.Боярского. - М.: Иностранная литература, 1963. - 388с.

128. Мунерман В.И. Абстрактные алгебраические машины как технология программирования – Системы компьютерной математики и их приложения: материалы международной конференции /Министерство образования РФ; Смоленский гос. ун-т. – Смоленск: Изд-во СмолГУ, 2007. – Вып. 8. стр. 106-109.

129. Емельченков Е. П., Левин Н. А., Мунерман В. И. Алгебраический подход к оптимизации разработки и эксплуатации систем управления базами данных //Системы и средства информатики. – 2009. – Т. 19. – №. 2. – С. 114-137.

130. Мунерман В.И., Кавченков Д.Е. Подход к созданию универсальной алгебраической машины для реализации алгебры многомерных матриц – Системы компьютерной математики и их приложения: материалы международной конференции /Министерство образования РФ; Смоленский гос. ун-т. – Смоленск: Изд-во СмолГУ, 2024. – Вып. 25. стр. 113-118.

131. Гендель Е.Г. Левин Н.А. Оптимизация технологии обработки информации в АСУ. – М.: Статистика, 1977 г. – 231 с.

132. Мартин Дж. Организация баз данных в вычислительных системах. – М.:

Мир, 1980 г. – 662 с.

133. Дейт К. Дж. Введение в системы баз данных. Восьмое издание. – Вильямс, 2008. – 1328 стр., с ил.; ISBN 978-5-8459-0788-2, 0-321-19784-4.

134. Джадд Д. Р. Работа с файлами. – М.: Мир, 1975. – 144 с.

135. Codd E.F. Providing OLAP for end-user analysis: An IT mandate. //ComputerWorld, 1993.

136. Codd E. F., Codd S. B., Salley C. T. Providing OLAP to User-Analysts: An IT Mandate, Arbor Software Corp. Papers, 1996.

137. Whitehorn M., Zare R., Pasumansky M. Fast track to MDX. – Springer Science & Business Media, 2007. ISBN-10: 1846281741 | 310 pages.

138. Hasan K. M. A., Tsuji T., Higuchi K. An efficient implementation for MOLAP basic data structure and its evaluation //Advances in Databases: Concepts, Systems and Applications: 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007. Proceedings 12. – Springer Berlin Heidelberg, 2007. – С. 288-299.

139. Dechter R. Constraint processing. – Morgan Kaufmann, San Francisco, 2003.

140. Buscemia M. G., Montanarib U. A survey of constraint-based programming paradigms. // Computer Science Review, Vol. 2, Issue 3, Dec. 2008, pp. 137–141. Русский перевод: Сергей Кузнецов 24.04.2003. Открытые системы, #04/2003

141. Stapp L. Axiomatic approach to the system of files //Workshop on Logic of Programs. – Berlin, Heidelberg: Springer Berlin Heidelberg, 1980. – С. 270-294.

142. Andrews G. R., Reitman R. P. An axiomatic approach to information flow in programs //ACM Transactions on Programming Languages and Systems (TOPLAS). – 1980. – Т. 2. – №. 1. – С. 56-76.

143. Chen Y. T., Tanik M. M. An axiomatic approach of software functionality measure //The Third International Workshop on Rapid System Prototyping. – IEEE Computer Society, 1992. – С. 181-187.

144. Hintersteiner J. D., Nain A. S. Integrating software into systems: an axiomatic design approach //The Third International Conference on Engineering Design and Automation. – 1999. – С. 1-4.

145. Мунерман В. И. Опыт массовой обработки данных в облачных системах (на примере Windows Azure) //Системы высокой доступности. – 2014. – Т. 10. – №. 2. – С. 3-8.

146. Мунерман В. И. Построение архитектур программно-аппаратных комплексов для повышения эффективности массовой обработки данных //Системы высокой доступности. – 2014. – Т. 10. – №. 4. – С. 3-16.

147. Захаров В.Н., Мунерман В.И. Алгебраический подход к формализации параллелизма данных. – Современные информационные технологии и ИТ-образование. Москва: Т.12, №1, 2016. – С. 72-79.

148. Бурбаки Н. Теория множеств. – М.: Мир – 1965.

149. Мендельсон Э. Введение в математическую логику. М: Наука – 1971.

150. Мунерман В. И. Аксиоматический метод формализации массовой обработки данных в системах высокой доступности //Системы высокой доступности. – 2017. – Т. 13. – №. 2. – С. 56-62.

151. Munerman V., Munerman D. An axiomatic approach to the data models formalization for mass data processing //2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2020. – С. 1996-2000.

152. Левин Н. А., Мунерман В. И. Алгебраический подход к оптимизации обработки информации //Системы и средства информатики. – 2005. – Т. 15. – №. 2. – С. 282-297.

153. Захаров В. Н., Мунерман В. И. Синтез и оптимизация запросов в системах массовой обработки данных //Системы компьютерной математики и их приложения. – 2015. – №. 16. – С. 77-82.

154. Мунерман В. И., Мунерман Д. В. Оптимизация процессов и операций массовой обработки данных //Системы компьютерной математики и их приложения. – 2020. – №. 21. – С. 172-178.

155. Непомнящий В. А. Верификация программ обработки файлов на языке Паскаль //Программирование. – 1981. – №. 2. – С. 34-43.

156. Непомнящий В. А., Рякин О. М. Прикладные методы верификации про-

грамм. – Государственное унитарное предприятие научно-техническое издательство" Радио и связь", 1988.

157. Ануреев И.С., Марьясов И.В., Непомнящий В.А. Верификация С-программ на основе смешанной аксиоматической семантики // Моделирование и анализ информационных систем. 2010. Том 17, № 3. С. 5–28.

158. Мунерман В.И., Мунерман Д.В. Аксиоматический метод доказательства соответствия формализованных в различных моделях данных запросов //Современные информационные технологии и ИТ-образование. – 2023. – Т. 19. – №. 4.

159. Ефимов С.С. Обзор методов распараллеливания алгоритмов решения некотоых задач вычислительной дискретной математики / Журнал «Математические структуры и моделирование». – Омск: Омск. гос. техн. ун-т, 2007. – Вып. 17. – 125 с.

160. Габасов Р., Кириллова Ф.М. Основы динамического программирования – Мн.: Изд-во БГУ, 1975. – 264 с.

161. Габасов Р. Методы оптимизации: пособие //Минск: Четыре четверти. – 2011. – 472 с.: ил.

162. Левин Н. А., Мунерман В. И. Алгебраический подход к оптимизации обработки информации //Системы и средства информатики. – 2005. – Т. 15. – №. 2. – С. 282-297.

163. Захаров В. Н., Мунерман В. И. Синтез и оптимизация запросов в системах массовой обработки данных //Системы компьютерной математики и их приложения. – 2015. – №. 16. – С. 77-82.

164. Мунерман В. И., Мунерман Д. В. Оптимизация процессов и операций массовой обработки данных //Системы компьютерной математики и их приложения. – 2020. – №. 21. – С. 172-178.

165. Fletcher P., Hoyle H., Patty C. W. Foundations of Discrete Mathematics. – Boston: PWS-KENT Pub. Co., 1991 – p. 781, ISBN 0-53492-373-9.

166. Lidl R., Pilz G. – Applied abstract algebra, 2nd edition, Undergraduate Texts in Mathematics, Springer, 1998. – p. 488, ISBN 0-387-98290-6.

167. Gross J. L., Yellen J. Handbook of Graph Theory, Discrete Mathematics and Its Applications. – CRC Press, 2004. – p. 779, ISBN 9780203490204.

168. Валях Е. Последовательно-параллельные вычисления. Пер. с англ . – М.: Мир, 1985. – 456 с., ил.

169. Писсанецки С. Технология разреженных матриц: Пер. с англ . – М.: Мир, 1988. – 410 с., ил. – ISBN 5-03-000960-4.

170. Малышев А.В. Паралеллизация умножения матриц. – Электронное научное периодическое издание "Электроника и информационные технологии", 2(4), 2008 г. – fetmag.mrsu.ru.

171. Choi, J., J. J. Dongarra, and D. W. Walker PUMMA: Parallel Universal Matrix Multiplication Algorithms on distributed memory concurrent computers," Concurrency: Practice and Experience, Vol 6(7), 543-570, 1994.

172. Choi J. A fast scalable universal matrix multiplication algorithm on distributed-memory concurrent computers //Proceedings 11th International Parallel Processing Symposium. – IEEE, 1997. – С. 310-314.

173. Van De Geijn R. A., Watts J. SUMMA: Scalable universal matrix multiplication algorithm //Concurrency: Practice and Experience. – 1997. – Т. 9. – №. 4. – С. 255-274.

174. Захаров В.Н., Мунерман В.И. Параллельная реализация обработки интенсивно используемых данных на основе алгебры многомерных матриц // Аналитика и управление данными в областях с интенсивным использованием данных: XVII Международная конференция DAMDID/RCDL'2015 (Обнинск, 13 - 16 октября 2015 года, Россия): Труды конференции - Обнинск: ИАТЭ НИЯУ МИФИ, 2015, с. 217 - 223. ISBN 978-5-9530-0398-.

175. Мунерман В. И., Парфенов Н. В. Анализ параллельного алгоритма умножения многомерных матриц //Системы компьютерной математики и их приложения. – 2016. – №. 17. – С. 68-70.

176. Goncharov E., Munerman V., Yakovlev G. Software and Hardware Complex for Calculating Convolutions by Methods Multidimensional Matrix Algebra //2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engi-

neering (ElConRus). – IEEE, 2021. – С. 2176-2180.

177. Munerman V. I., Munerman D. V. Обобщение одного алгоритма параллельного умножения матриц в алгебре многомерных матриц //Современные информационные технологии и ИТ-образование. – 2022. – Т. 18. – №. 3. – С. 566-577.

178. Shneider D., DeWitt D. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment // Proceeding of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon. – New York, NY 10036, USA, ACM Press, 1989. – p. 110-121.

179. Wolf, J.L., Watson T. J., Yu, P.S., Turek J., Dias, D.M. A parallel hash join algorithm for managing data skew. – Parallel and Distributed Systems, IEEE Transactions, v. 4, Issue: 12, 1993. – p.1355 – 1371.

180. Костенецкий П.С. Обработка запросов на кластерных вычислительных системах с многоядерными ускорителями // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2012. № 47(306). Вып. 2. С. 59–67.

181. Приказчиков С.О., Костенецкий П.С. Применение графических ускорителей для обработки запросов над сжатыми данными в параллельных системах баз данных // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4. № 1. С. 64-70.

182. Lang H., Leis, V., Albutiu, M. C., Neumann, T., Kemper, A. Massively parallel NUMA-aware hash joins //In Memory Data Management and Analysis: First and Second International Workshops, IMDM 2013, Riva del Garda, Italy, August 26, 2013, IMDM 2014, Hongzhou, China, September 1, 2014, Revised Selected Papers. – Springer International Publishing, 2015. – С. 3-14.

183. Barber R., Lohman G., Pandis I., Raman V., Sidle R., Attaluri G., Chainani N., Lightstone S., Sharpe D. Memory-efficient hash joins. – Journal Proceedings of the VLDB Endowment, v. 8, Issue 4, 2014. – p. 353-364.

184. Zaqout F., Abbas M., Hosam A. S. Indexed Sequential Access Method (ISAM): A Review of the Processing Files //2011 UKSim 5th European Symposium on

Computer Modeling and Simulation. – IEEE, 2011. – С. 356-359.

185. Аверченков В. И. и др. Анализ и некоторая классификация методов доступа к данным //Известия Волгоградского государственного технического университета. – 2014. – №. 12. – С. 44-51.

186. Martelo S., Toth P. Knapsack problems. Algorithms and computer implementations. – John Wiley & Sons, Chichester. New-York. Brisbane. Toronto. Singapore, 1990. – p. 306.

187. Puchinger J., Raidl G. R., Pferschy U. The multidimensional knapsack problem: Structure and algorithms //INFORMS Journal on Computing. – 2010. – Т. 22. – №. 2. – С. 250-265.

188. Мунерман В. И., Мунерман Д. В. Алгебраический подход к построению программно-аппаратных комплексов для повышения эффективности массовой обработки данных //Современные информационные технологии и ИТ-образование. – 2015. – Т. 2. – №. 11. – С. 391-396.

189. Мунерман В. И., Мунерман Д. В. Параллельная реализация операций обработки файлов //Современные информационные технологии и ИТ-образование. – 2016. – Т. 12. – №. 2. – С. 84-90.

190. Мунерман В. И., Мунерман Д. В. Один метод реализации симметричного горизонтального распределения данных //Системы компьютерной математики и их приложения. – 2017. – №. 18. – С. 97-100.

191. Мунерман В. И., Мунерман Д. В. Параллельная реализация симметричного горизонтального распределения данных на основе сетевых технологий //Современные информационные технологии и ИТ-образование. – 2017. – Т. 13. – №. 3. – С. 38-43.

192. Мунерман В. И. Реализация параллельной обработки данных в облачных системах //Современные информационные технологии и ИТ-образование. – 2017. – Т. 13. – №. 2. – С. 57-63.

193. Munerman V., Munerman D., Samoilova T. The Heuristic Algorithm For Symmetric Horizontal Data Distribution //2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). – IEEE, 2021. – С.

2161-2165.

194. Иванов М. С., Мунерман В. И. Генерация хранимых процедур для реализации алгоритма бустрофедона //Системы компьютерной математики и их приложения. – 2019. – №. 20-1. – С. 157-161.

195. Постановление Правительства Российской Федерации от 28.12.2022 № 2461 [Электронный ресурс]. URL: http://publication.pravo.gov.ru/Document/View/0001202212300083/(Дата обращения: 26.03.2024)

196. Многопроцессорные системы с реконфигурируемой архитектурой. Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров [Электронный ресурс]. URL: http://superevm.ru/index.php?page=hardware/ (Дата обращения: 26.03.2023).

197. Darema F. The spmd model: Past, present and future //Recent Advances in Parallel Virtual Machine and Message Passing Interface: 8th European PVM/MPI Users' Group Meeting Santorini/Thera, Greece, September 23–26, 2001 Proceedings 8. – Springer Berlin Heidelberg, 2001. – С. 1-1.

198. Database Data Warehousing Guide. ORACLE* Help Center. [Электронный ресурс]. URL: https://docs.oracle.com/cd/B19306_01/server.102/b14223/usingpe.htm/ (Дата обращения: 26.03.2023).

199. Exploring SQL Server 2014 SELECT INTO Parallelism. Solarwinds Sentry-One . [Электронный ресурс]. URL: http://sqlperformance.com/2013/08/t-sql-queries/parallel-select-into/ (Дата обращения: 26.03.2023).

200. Гайдаенко Т. И., Хайлов И. К. Архитектурные особенности потокового мультипроцессорного вычислителя системы «ВСАРР» //Системы и средства информатики. – 2006. – №. 16. – С. 404-417.

201. Провоторова А. О. Моделирование параллельной системы баз данных на вычислительной системе «ВСАРР» //Системы и средства информатики. – 2006. – №. 16. – С. 418-430.

202. Макаров Д. И., Мунерман В. И. Анализ эффективности обработки боль-

ших объемов данных на вычислительных комплексах массового параллелизма //Системы компьютерной математики и их приложения. – 2013. – №. 14. – С. 85-87.

203. Мунерман В. И., Надэлин А. А. Параллельная реализация операции JOIN средствами технологии CUDA //Системы компьютерной математики и их приложения. – 2016. – №. 17. – С. 66-67.

204. Bast, H.: Car or public transport – two worlds. In: Efficient Algorithms, LNCS, vol. 5760, pp. 355–367. Springer (2009).

205. Беляков С.Л., Коломийцев Я.А., Розенберг И.Н., Савельева М.Н., Модель решения задачи маршрутизации в интеллектуальной геоинформационной системе. Известия Южного федерального университета. Технические науки, 2011, Том 118, Вып.5, с.113-119.

206. Мунерман В. И., Самойлова Т. А. Параллельная реализация решения оптимизационных задач средствами баз данных //Системы высокой доступности. – 2015. – Т. 11. – №. 1. – С. 18-22.

207. Miller J.A., Ramaswamy L., Kochut K.J., Fard A. Research Directions for Big Data Graph Analytics, IEEE International Congress on BigData, 2015, 785-794.

208. Марголис Б.И., Музанна М.М. Синтез магистральных телекоммуникационных сетей //Программные продукты и системы. Тверь, 2014, № 1 (105), С. 162-168.

209. Давыденко В. А., Ромашкина Г. Ф., Чуканов С. Н. Моделирование социальных сетей //Вестник Тюменского государственного университета.—2005.—№ 1. – 2005.

210. Amati V., Lomi A., Mira A. Social network modeling //Annual Review of Statistics and Its Application. – 2018. – Т. 5. – №. 1. – С. 343-369.

211. Семенов Ю.Н., Семенова О.С. Применение методов кластеризации при организации международных перевозок грузов. Вестник КузГТУ.- 2016, -№6, с.201- 205

212. Masum K., Faruque F, Shahjalal M., Sarker H. Solving the Vehicle Routing Problem using Genetic Algorithm. (IJACSA)International Journal of Advanced

Computer Science and Applications. Vol.2. No.7, 2011, 126-131.

213. Емельченков, Е. П., Мунерман, В. И., Мунерман, Д. В., Самойлова, Т. А. Один метод построения циклов в графе //Современные информационные технологии и ИТ-образование. – 2021. – Т. 17. – №. 4. – С. 814-823.

214. Мунерман В. И., Самойлова Т. А. Алгебраический подход к алгоритмизации задач маршрутизации //Системы высокой доступности. – 2018. – Т. 14. – №. 5. – С. 50-56.

215. Морозов С. А., Мунерман В. И., Симаков В. А. Экспериментальный анализ многомерно-матричного подхода к построению маршрутов в графе //Известия высших учебных заведений. Электроника. – 2022. – Т. 27. – №. 5. – С. 676-686.

216. Morozov S. A., Munerman V. I., Simakov V. A. Experimental Analysis of the Multidimensional-Matrix Approach to Construct Routes in a Graph //Russian Microelectronics. – 2023. – Т. 52. – №. 7. – С. 716-721.

217. Морозов С. А., Мунерман В. И. Параллельная реализация алгоритма построения всех маршрутов в графе средствами реляционной алгебры //Современные информационные технологии и ИТ-образование. – 2023. – Т. 19. – №. 4.

218. Houtsma M., Swami A. Set-oriented mining of association rules. Research Report RJ 9567, IBM Almaden Research Center, San Jose, California, October 1993.

219. Agrawal R., Srikant R. Fast Discovery of Association Rules. – Proc. of the 20th International Conference on VLDB, Santiago, Chile, September 1994.

220. Srikant R., Agrawal R. Mining Generalized Association Rules. – Proc. of the 21th International Conference on VLDB, Zurich, Switzerland, 1995.

221. Pol U. Design and Development of Apriori Algorithm for Sequential to concurrent mining using MPI / U. Pol // International journal of Computers & Technology. – 2013. – Т. 10. – № 7. – С. 1785–1790.

222. Salim M., Yao X. Evolving SQL Queries for Data Mining. – Lecture Notes in Computer Science, 2002, 2412. – p. 62–67.

223. R. Agrawal, T. Imielinski, A. Swami. 1993. Mining Associations between Sets

of Items in Massive Databases. In Proc. of the 1993 ACM-SIGMOD Int'l Conf. on Management of Data, 207-216.

224. R. Srikant, R. Agrawal. Mining quantitative association rules in large relational tables. In Proceedings of the ACM SIGMOD Conference on Management of Data, Montreal, Canada, June 1996.

225. Речкалов Т.В. Подход к интеграции интеллектуального анализа данных в реляционную СУБД на основе генерации текстов хранимых процедур. – Вестник Южно-Уральского государственного университета. Серия «Вычислительная математика и информатика», том 2, №1, издательство: Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск), 2013. – с. 114-121.

226. Sidló, C.I., Lukács, A. Shaping SQL-based frequent pattern mining algorithms (Revised Selected and Invited Papers). – Knowledge Discovery in Inductive Databases: 4th International Workshop, KDID 2005, Springer, Heidelberg. – p. 188–201.

227. Python Package Index. [Электронный ресурс]. URL: https://pypi.python.org/pypi/ (Дата обращения: 26.03.2023).

228. Захаров В. Н., Мунерман В. И., Самойлова Т. А. Параллельные методы вывода ассоциативных правил в технологиях in-database и in-memory //Конвергентные когнитивно-информационные технологии: сб. тр. науч. конф. М. – 2017. – С. 219-225.

229. Haynes D., Ray S., Manson S. M., Soni A. High performance analysis of big spatial data //2015 IEEE International Conference on Big Data (Big Data). – IEEE, 2015. – С. 1953-1957.

230. Monga V., Evans B. L. Perceptual image hashing via feature points: performance evaluation and tradeoffs //IEEE transactions on Image Processing. – 2006. – Т. 15. – №. 11. – С. 3452-3465.

231. Chamoso P., Rivas A., Sanchez-Torres R., Rodriguez, S. Social computing for image matching //PloS one. – 2018. – Т. 13. – №. 5. – С. e0197576.

232. Кирикова А. В., Мунерман В. И., Самойлова Т. А. Реализация поиска изображений в базах данных //Системы компьютерной математики и их прило-

жения. – 2019. – №. 20-1. – С. 167-172.

233. Zakharov V., Kirikova A., Munerman V., Samoilova, T. Architecture of Software-Hardware Complex for Searching Images in Database //2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2019. – С. 1735-1739.

234. Kirikova A., Mironov A., Munerman V. The Method of Composition Hash-functions for Optimize a Task of Searching Images in Dataset //2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2020. – С. 1983-1986.

235. Stallings W. Cryptography and Network Security: Principles and Practice. – Pearson, 2011. – 711 p.

236. Qasem M. H., Qatawneh M. Parallel Hill Cipher Encryption Algorithm //International Journal of Computer Applications. – 2018. – V. 179. – #19. – С. 16-24.

237. Ismail I. A., Amin M., Diab H. How to repair the Hill cipher //Journal of Zhejiang University-Science A. – 2006. – V. 7. – #12. – P. 2022-2030.

238. Parmar N. B., Bhatt K. R. Hill cipher modifications: A detailed review //International Journal of Innovative Research in Computer and Communication Engineering. – 2015. – V. 3. – # 3. – P. 1467-1474.

239. Maxrizal M., Prayanti B.D.A. A New Method Of Hill Cipher: The Rectangular Matrix As The Private Key. – 2nd International Conference on Science and Technology for Sustainability Proceeding. – 2016. – V. 2. – P. 81-83.

240. Гончаров Е. И., Мунерман В. И., Самойлова Т. А. Выбор параметров многомерных матриц для обобщенного алгоритма шифрования Хилла //Системы компьютерной математики и их приложения. – 2019. – №. 20-1. – С. 111-116.

241. Мунерман В. И., Самойлова Т. А. Реализация алгоритма шифрования Хилла на основе алгебры многомерных матриц //Системы высокой доступности. – 2019. – Т. 15. – №. 1. – С. 21-27.

242. Ильин П. Л., Мунерман В. И. Рекурсивное вычисление детерминанта многомерной матрицы //Системы компьютерной математики и их приложения.

– 2019. – №. 20-1. – С. 162-167.

243. Патентом на полезную модель RUS 82355 12.08.2008/ Система представления данных в базе данных Сергеев В.П., Гайдаенко Т.И., Левин Н.А., Мунерман В.И., Оздемир С.М., Провоторова А.О., Ширай А.Е.

244. Патентом № 2755568 Российская Федерация, МПК G06F 16/2455. Способ параллельного выполнения операции JOIN при обработке больших структурированных высокоактивных данных: №2020124733: заявл. 26.07.2020: опубл. 17.09.2021 / Мунерман В. И., Синявский Ю. В., Чукляев И. Л., Чукляев Е. И. – 10 с.

Приложение 1. Патент на полезную модель RUS 82355

РОССИЙСКАЯ ФЕДЕРАЦИЯ

(19) **RU** (11) **82 355** (13) **U1**

(51) МПК
G06F 17/30 (2006.01)

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ,
ПАТЕНТАМ И ТОВАРНЫМ ЗНАКАМ

(12) ОПИСАНИЕ ПОЛЕЗНОЙ МОДЕЛИ К ПАТЕНТУ

Статус: не действует (последнее изменение статуса: 02.07.2021)
Пошлина: учтена за 8 год с 13.08.2015 по 12.08.2016. Патент перешел в общественное достояние.

(21)(22) Заявка: **2008132952/22, 12.08.2008**

(24) Дата начала отсчета срока действия патента:
12.08.2008

(45) Опубликовано: 20.04.2009 Бюл. № **11**

Адрес для переписки:
119296, Москва, а/я 98, Л.Г. Багяну

(72) Автор(ы):
Сергеев Виктор Петрович (RU),
Гайдаенко Татьяна Ивановна (RU),
Левин Нисон Абрамович (RU),
Мунерман Виктор Иосифович (RU),
Оздемир Светлана Марковна (RU),
Провоторова Анна Олеговна (RU),
Ширай Александр Евгеньевич (RU)

(73) Патентообладатель(и):
Сергеев Виктор Петрович (RU)

(54) **СИСТЕМА ПРЕДСТАВЛЕНИЯ ДАННЫХ В БАЗЕ ДАННЫХ**

Приложение 2. Патент № 2755568

РОССИЙСКАЯ ФЕДЕРАЦИЯ

(19) **RU** (11) **2 755 568**(13) **C1**

(51) МПК
G06F 16/2455 (2019.01)

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

(12) ФОРМУЛА ИЗОБРЕТЕНИЯ К ПАТЕНТУ РОССИЙСКОЙ ФЕДЕРАЦИИ

(52) СПК
G06F 16/2455 (2021.08)

(21)(22) Заявка: **2020124733, 26.07.2020**

(24) Дата начала отсчета срока действия патента:
26.07.2020

Дата регистрации:
17.09.2021

Приоритет(ы):
(22) Дата подачи заявки: **26.07.2020**

(45) Опубликовано: **17.09.2021** Бюл. № **26**

Адрес для переписки:
214004, г. Смоленск, ул. Ново-Киевская 2 кв.88, Синявский Юрий Владимирович

(73) Патентообладатель(и):
Смоленская общественная организация содействия науке и образованию «Региональный союз ученых» (RU)

(56) Список документов, цитированных в отчете о поиске: **US 8099409 B2, 17.01.2012. US 9665624 B2, 30.05.2017. US 8914354 B2, 16.12.2014. US 10380112 B2, 13.08.2019. US 6957210 B1, 18.10.2005. WO 2019/018271 A1, 24.01.2019. US 10268639 B2, 23.04.2019. WO 1997/011432 A1, 27.03.1997. US 2018/0004810 A1, 04.01.2018.**

(54) **Способ параллельного выполнения операции JOIN при обработке больших структурированных высокоактивных данных**

RU 2755568 C1

(57) Формула изобретения

Способ параллельного выполнения операции JOIN при обработке больших структурированных высокоактивных данных, заключающийся в том, что в компьютерно-реализуемой системе манипулирования данными, использующей язык SQL, после получения хост-процессором обеих таблиц-операндов и условий отбора строк в операции производится распределение строк таблиц-операндов между множеством модулей обработки в форме таблиц-фрагментов с последующим параллельным выполнением операции SQL JOIN в модулях обработки над этими фрагментами таблиц-операндов с одновременным объединением результатов, отличающийся тем, что распределению строк таблиц-фрагментов между множеством модулей обработки предшествует осуществляемое хост-процессором формирование для каждой таблицы-операнда индексной таблицы, в которой каждая строка содержит значение ключа операции SQL JOIN и значение количества строк в данной таблице-операнде, соответствующее этому ключу, выполнение операции пересечения сформированных индексных таблиц с вычислением произведения частот использования ключей операции SQL JOIN в таблицах-операндах, упорядочивание полученной сводной таблицы метаданных по убыванию значений величины произведения частот использования ключей, отбор строк таблиц-операндов в таблицы-фрагменты осуществляется по значениям ключей, соответствующих строкам сводной таблицы метаданных, таким образом, что каждая пара таблиц-фрагментов в одном модуле

RU 2755568 C1

обработки содержит только те строки таблиц-операндов, которые имеют одинаковое значение ключа.

RU 2755568 C1

PCT

ЗАЯВЛЕНИЕ

Нижеподписавшийся просит рассматривать настоящую международную заявку в соответствии с Договором о патентной кооперации

Заполняется Получающим ведомством

Номер международной заявки: **PCT/RU2021/000480**

Дата международной подачи: **02 НОЯБРЯ 2021 (02.11.2021)**

Наименование Получающего ведомства и знаки: «Международная заявка PCT»: **RO/RU МЕЖДУНАРОДНАЯ ЗАЯВКА PCT PCT INTERNATIONAL APPLICATION**

№ дела заявителя или агента *(по желанию) (максимум 25 знаков)*: RSU_001-PCT

Графа I НАЗВАНИЕ ИЗОБРЕТЕНИЯ

Способ оптимизации параллельного выполнения операции JOIN при обработке больших структурированных высокоактивных данных

ФИПС
02 НОЯ 2021
ВХОД. ОТД. № 17

Графа II ЗАЯВИТЕЛЬ ☐ Данное лицо является также изобретателем

Имя и адрес: *(Фамилия указывается перед именем, для юридического лица - полное уставное наименование. Адрес должен включать почтовый индекс и название страны. Если государство местожительства внизу не будет указано, то таковым будет считаться страна указанного в данной графе адреса)*

Смоленская общественная организация содействия науке и образованию «Региональный союз ученых», Заречная ул., д. 27, Смоленск, Смоленская область, 214010, РФ
Smolensk public organization for the promotion of science and education "Regional union of scientists", Zaretchnaya ul., d. 27, Smolensk, Smolenskaya region, 214010, RU

Телефон №: (+7-920)3076962

Телефакс №:

Регистрационный № заявителя в Ведомстве:

E-mail разрешение: Пометка одного из боксов ниже позволяет Получающему ведомству, Международному поисковому органу, Международному бюро и Органу международной предварительной экспертизы, по их желанию, использовать указанный в данной графе e-mail адрес для отправки по этому e-mail адресу уведомлений, подготовленных в отношении данной международной заявки.

☒ в качестве предварительных копий, вслед за которыми высылаются уведомления на бумаге; или ☐ только в электронной форме (никакие уведомления на бумаге направляться не будут).

E-mail адрес: smolrsu-pp@mail.ru

Государство *(т.е. страна)* гражданства: Россия (RU)

Государство *(т.е. страна)* местожительства: Россия (RU)

RO/RU

Данное лицо является заявителем для: ☒ Всех Указанных государств ☐ Государств, названных в дополнительной графе

Графа III ДРУГИЕ ЗАЯВИТЕЛИ И/ИЛИ (ДРУГИЕ) ИЗОБРЕТАТЕЛИ

☒ Другие заявители и/или (другие) изобретатели указаны на листе для продолжения

Графа IV АГЕНТ ИЛИ ОБЩИЙ ПРЕДСТАВИТЕЛЬ; ИЛИ АДРЕС ДЛЯ ПЕРЕПИСКИ

Указанное ниже лицо настоящим назначается (назначено) представлять интересы заявителя(ей) в компетентных международных органах в качестве: ☐ агента ☐ общего представителя

Имя и адрес: *(Фамилия указывается перед именем, для юридического лица - полное уставное наименование. Адрес должен включать почтовый индекс и название страны)*

Телефон №:

Телефакс №:

Регистрационный № агента в Ведомстве:

E-mail разрешение: Пометка одного из боксов ниже позволяет Получающему ведомству, Международному поисковому органу, Международному бюро и Органу международной предварительной экспертизы, по их желанию, использовать указанный в данной графе e-mail адрес для отправки по этому e-mail адресу уведомлений, подготовленных в отношении данной международной заявки.

☒ в качестве предварительных копий, вслед за которыми высылаются уведомления на бумаге; или ☐ только в электронной форме (никакие уведомления на бумаге направляться не будут).

E-mail адрес: smolrsu-pp@mail.ru

☒ **Адрес для переписки:** Пометить этот бокс, если агент или общий представитель не назначаются (не назначены), а указанный выше адрес используется только как специальный адрес для переписки.

Приложение 3. Свидетельства о регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2020614270

Программа для обработки распределенных больших объемов данных в вычислительных сетях рабочих станций

Правообладатель: *Смоленская общественная организация содействия науке и образованию "Региональный союз ученых" (RU)*

Авторы: *Мунерман Виктор Иосифович (RU), Самойлов Михаил Юрьевич (RU), Макаров Дмитрий Игоревич (RU)*

Заявка № 2019667738
Дата поступления 27 декабря 2019 г.
Дата государственной регистрации
в Реестре программ для ЭВМ *27 марта 2020 г.*

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2020613833

Программа управления обработкой распределенных больших объемов данных для стоечных серверов и дата-центров

Правообладатель: *Смоленская общественная организация содействия науке и образованию "Региональный союз ученых" (RU)*

Авторы: *Мунерман Виктор Иосифович (RU), Макаров Дмитрий Игоревич (RU), Самойлов Михаил Юрьевич (RU)*

Заявка № 2019667860
Дата поступления 30 декабря 2019 г.
Дата государственной регистрации
в Реестре программ для ЭВМ *23 марта 2020 г.*

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

Приложение 4. Акт внедрения результатов диссертационной работы

УТВЕРЖДАЮ
Директор по НИОКР
АО «Т-Платформы»

А.Е. Березко

«20» 06 2018 г.

А К Т
внедрения результатов диссертационной работы
Мунермана Виктора Иосифовича
на соискание ученой степени доктора технических наук

Рассмотрев основные результаты, выводы и рекомендации диссертационной работы Мунермана Виктора Иосифовича на соискание ученой степени доктора технических наук по специальности 05.13.15 «Вычислительные машины, комплексы и компьютерные сети»,

комиссия в составе:

председателя
– руководителя Департамента сопровождения НИОКР Вострова Дмитрия Викторовича;

членов комиссии:
– старшего руководителя проектов Васильева Александра Олеговича;
– старшего руководителя проектов Ивановской Елены Владимировны

установила следующее:

Разработанное научно-методическое обеспечение, состоящее в:
– многомерно-матричной и теоретико-множественной (файловой) моделях данных и доказательстве их соответствия;
– методах синтеза и оптимизации процессов обработки данных;
– методах установления соответствия между моделями данных и архитектурами программно-аппаратных комплексов;
– рекомендации по построению программно-аппаратных комплексов

использованы *при проектировании и разработке линейки серверов в форм-факторе «блейд» для высокопроизводительных вычислений и дата-центров на базе процессоров x86 и отечественных процессоров семейства «Байкал»*

для повышения их производительности и уровня соответствия решаемым с их использованием задачам.

ВЫВОДЫ:

Указанные научно-методические результаты и выработанные рекомендации **позволили** *дополнить научно-технический задел по разработке базовых технологий производства приоритетных электронных компонентов и радиоэлектронной аппаратуры в рамках подпрограммы «Развитие производства вычислительной техники» Государственной программы Российской Федерации «Развитие электронной и радиоэлектронной промышленности на 2013-2025 годы».*

Председатель комиссии:
Руководитель Департамента
сопровождения НИОКР Д.В. Востров

Члены комиссии:
Старший руководитель проектов А.О. Васильев

Старший руководитель проектов Е.В. Ивановская

« 20 » 06 2018 г.