

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение
высшего образования «Казанский национальный исследовательский
технический университет им. А.Н. Туполева-КАИ»
Институт компьютерных технологий и защиты информации

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ по дисциплине
Б1.О.25 «Аппаратные средства вычислительной техники»

I. СИСТОЛИЧЕСКИЕ СТРУКТУРЫ	8
Введение. ПОДХОД К СИНТЕЗУ ОПЕРАЦИОННЫХ ЛОГИКО-ЗАПОМИНАЮЩИХ СРЕД	8
Занятие 1. ПЕРЕСТРАИВАЕМАЯ СРЕДА ЧИСЛОВОГО ПОИСКА	10
Лекция 1. Синтез среды числового поиска	10
Деление массива на три части. Поиск минимума	
Практика 1. Моделирование 1	13
Демонстрация. Тестирование.	
Занятие 2. СРЕДА ПОИСКА МИНИМАЛЬНОГО ПОКРЫТИЯ	21
Лекция 2. Организация операционной матрицы	21
Практика 2. Моделирование 2	26
Демонстрация. Тестирование.	
Занятие 3. СРЕДА ОДНОТАКТНОГО РАСПОЗНАВА- НИЯ	32
Лекция 3. Матрица распознавания	32
Практика 3. Моделирование 3	35
Демонстрация. Тестирование.	
Занятие 4. МНОГОТАКТНОЕ РАСПОЗНАВАНИЕ. СПЕЦПРОЦЕССОР-ИДЕНТИФИКАТОР	40
Лекция 4. Многотактный алгоритм и его реализация	40
Необходимость многотактного распознавания. Парал- лельный алгоритм. Структура спецпроцессора. Функцио- нирование комплекса.	
Практика 4. Моделирование 4	50
Необходимая справка. Способ задания матриц.	

Одиночное распознавание. Множественное распознавание.

ЛИТЕРАТУРА К РАЗДЕЛУ I 59

II. ПАРАЛЛЕЛЬНЫЕ СУБД 60

**Введение. РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ.
ОСОБЕННОСТИ ПАРАЛЛЕЛЬНЫХ СУБД ... 60**

Реляционная модель данных 60

Операции над данными (реляционная алгебра) 63
Операции обработки кортежей. Операции обработки отношений

Свойства параллельных систем 66

Аппаратная архитектура систем баз данных 68

Занятие 5. СУБД MySQL Cluster 72

Лекция 5. Начальное знакомство 73

Архитектура 73

Принципы работы 74

Обеспечение надежности в MySQL Cluster 75
Синхронное зеркалирование. Обнаружение сбоя.
Восстановление системы.

Процедуры конфигурирования MySQL Cluster 78
Настройка MGM node. Настройка NDB node. Настройка API node.

Управление MySQL Cluster 81
Консоль управления ndb_mgm. Порядок запуска MySQL Cluster. Порядок останова кластера.

Практика 5. Лабораторная работа 83

Занятие 6. ПАРАЛЛЕЛЬНАЯ СУБД Clusterix 83

Лекция 6. Знакомство с принципами построения 83

Общее описание. Алгоритм работы системы. Хранение данных в системе. Команды управления кластером. Запуск кластера.

Практика 6. Лабораторная работа 95

Занятие 7. ОБРАБОТКА ЗАПРОСОВ В СУБД Clusterix.. 95

Лекция 7. Претрансляция и исполнение запросов 96

Формирование команд плана обработки запросов 96

Пример построения дерева обработки. Состав команд плана обработки запросов.

Параллельная обработка запроса 99

Параллельный алгоритм соединения. Выполнение сформированного плана обработки по тактам.

Практика 7. Лабораторная работа 105

ЛИТЕРАТУРА К РАЗДЕЛУ II 106

III. ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ЗАЩИЩЕННЫХ КАРТОГРАФИЧЕСКИХ БАЗ ДАННЫХ 107

Введение. НЕОБХОДИМЫЕ СВЕДЕНИЯ ИЗ КАРТОГРАФИИ 107

Географические координаты. Плоские прямоугольные координаты. Полярные координаты.

Занятие 8. ДВУМЕРНО-АССОЦИАТИВНЫЙ МЕХАНИЗМ ЗАЩИТЫ КАРТОГРАФИЧЕСКИХ ДАННЫХ 111

Лекция 8. Краткие сведения по алгоритму 111

Рассматриваемый пример. Стойкость шифра.

Практика 8. Лабораторная работа 115

Занятие 9. ПАРАЛЛЕЛЬНАЯ СУБД Security Map Cluster 116

Лекция 9. Описание процедур и функционирование 116

Формирование ЗКБД. Алгоритм шифрования картографической базы данных. Модуль управления. Реализованные процедуры. СУБД Security Map Cluster. Подсистема визуализации.

Практика 9. Лабораторная работа 126

ЛИТЕРАТУРА К РАЗДЕЛУ III 127

I. СИСТОЛИЧЕСКИЕ СТРУКТУРЫ

В этом разделе дается знакомство с принципами разработки специализированных процессорных матриц на двух уровнях иерархии представления параллельной системы: схемо- и системотехническом. Основное внимание уделяется синтезу операционных логико-запоминающих сред (ЛЗС) [5] различного функционального назначения как одной из разновидностей систолических структур [6]. Системотехнический уровень представлен примером реализации на основе таких сред обрабатывающей части специализированного матричного процессора-идентификатора. В совокупности это создает достаточно цельное представление об организации процессорных матриц.

Введение

ПОДХОД К СИНТЕЗУ ОПЕРАЦИОННЫХ ЛЗС

Определим операционную ЛЗС [1–3] как итеративную двумерную структуру рис 1.1, которая реализует заданное множество процедур независимо от размеров среды ($i=1,2, \dots, m; j=1,2, \dots, n$). Элемент (j, i) среды содержит ячейку памяти для хранения 1 бита исходной информации и автоматную часть. Признаки α_i, β_j являются общими для i -го столбца и j -ой строки. Они могут выполнять функции настройки, маскирования, разрешения считывания или записи, числовых разрядов, результатов анализа содержимого строк или столбцов и т.д.

В общем случае внешние сигналы элемента суть многокомпонентные наборы. Например $\alpha_i = (\alpha_i^1, \alpha_i^2, \dots, \alpha_i^k)$. Выходы элемента (j,i) являются автоматными функциями входов и содержимого $a_{j,i}$ ячейки памяти. Элементы могут иметь и отдельные выходы. Допускается коммутация шин по краям среды (рис. 1.1, пунктир). "Обрамление" среды составляют группы регистров, в которые помещаются исходные данные (признаки), маски и конечные результаты. В них может выполняться и дополнительная логическая обработка информации.

Суть используемого далее подхода к синтезу операционных логико-запоминающих сред заключается в следующем.

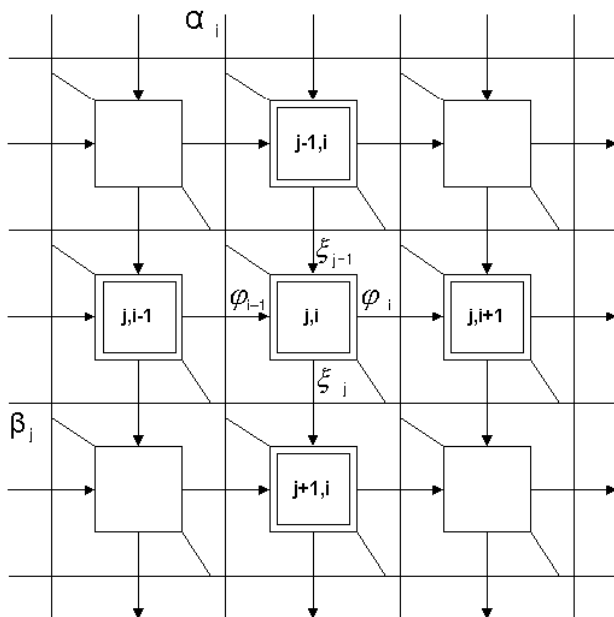


Рис. 1.1

В конечном итоге задача синтеза ЛЗС для реализации заданного набора процедур сводится к определению структуры связей, схемы элемента и значений сигналов на границах среды. Основное затруднение связано с тем, что неизвестна стандартная рецептура формулировки алгоритмов выполнения процедур на основе ЛЗС. В этом смысле любой подход к синтезу ЛЗС в целом эвристичен. Но если алгоритм приемлемо сформулирован и связи определены, то дальнейший процесс синтеза хорошо формализуется с помощью метода модифицированных таблиц переходов, развитого первоначально применительно к одномерным итеративным системам [4].

По определению в ЛЗС выполняется последовательно-однотипная обработка элементов информации по строкам и (или) столбцам. Подобную интерпретацию допускают [1–4], операции поиска числа в однородном информационном массиве по заданному набору признаков, распознавания каких-либо свойств такого массива

и преобразования кодов (суммирования, сдвигов и т.д.). Поэтому достаточным условием реализуемости произвольной процедуры на основе ЛЗС является возможность формулировки алгоритма ее выполнения в терминах выделенного класса операций.

Пусть такое условие выполнено. Тогда синтезируется своя среда для каждой из непересекающихся алгоритмических компонент (операций) путем интерпретации этой компоненты применительно к использованию ЛЗС совместно с методом модифицированных таблиц переходов. После этого либо организуется единая перестраиваемая среда объединением всех составляющих, либо создается обрабатывающий массив из нескольких специализированных сред.

На первых двух занятиях дается иллюстрация сформулированного подхода на примерах *структур информационного поиска и минимизации булевых функций*.

Заметим, что метод модифицированных таблиц переходов позволяет описать поведение как комбинационных, так и последовательностных итеративных систем. В последнем случае на этапе реализации элемента по такой таблице в цепь обратной связи необходимо включать синхронные D – триггеры (элементы задержки на такт) по одному на каждую внутреннюю переменную. Тем самым достаточно просто решается и проблема начальной установки автомата. Граничный набор действует постоянно. Такты разделяются подачей синхроимпульсов на D – триггер.

Занятие 1

ПЕРЕСТРАИВАЕМАЯ СРЕДА ЧИСЛОВОГО ПОИСКА

Лекция 1. Синтез среды числового поиска [1,2]

При структурной обработке информационных массивов наиболее часто встречаются задачи упорядочения элементов информации (сортировка), поиска по заданному признаку или совокупности признаков (ассоциативный поиск), выделения всех элементов, больших (меньших) заданного признака или расположенных в определенных границах и т. д. Их аппаратная реализация на основе ЛЗС может дать резкое повышение быстродействия по сравнению с программными методами.

Рассмотрим синтез перестраиваемой среды, которая реализует поиск минимальных (максимальных) чисел информационного массива, равных заданному, больших или меньших заданного, ближайшего большего (меньшего) по отношению к заданному. Поиск максимума равносильен поиску минимума среди обратных кодов чисел массива. Ближайшее большее - минимальный элемент среди всех чисел, превышающих заданное. Поэтому искомая среда является суперпозицией двух ЛЗС. Одна из них делит массив на три подмножества чисел - равных, больших и меньших заданного. Другая отыскивает минимальный элемент массива.

Деление массива на три части выполняется по следующему алгоритму. Поиск ведется слева направо (со старших разрядов). Если на предыдущей итерации строка отнесена к подмножествам ">" или "<", то обработка i-го бита не вносит изменений в результат. Если ранее строка была включена в подмножество "=", то результат i-ой итерации определяется соотношением между α_i^1 и $a_{j,i}$ (между i - ми разрядами признака и числа в j-й строке).

Боковые входы элемента (рис. 1.2)

$$\varphi_{i-1} \in \{S_0, S_1, S_2\}$$

Наборы S_0, S_1, S_2 отвечают случаям, когда левая по отношению к i-му разряду часть числа равна, больше или меньше левой части признака. Правый выход элемента определен модифицированной таблицей переходов (рис. 1.3). Выполняя кодировку состояний (рис. 1.4), получаем кодированную таблицу (рис. 1.5). Из этой таблицы:

$$\varphi_i^1 = \varphi_{i-1}^1 \vee \overline{\alpha_i^1} a_{j,i} \overline{\varphi_{i-1}^2}, \quad \varphi_i^2 = \varphi_{i-1}^2 \vee \alpha_i^1 \overline{a_{j,i}} \overline{\varphi_{i-1}^1}.$$

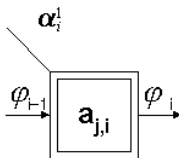


Рис. 1.2

		$\alpha_i^1 a_{j,i}$				φ_i
		00	01	11	10	
φ_{i-1}	S_0	S_0	S_1	S_0	S_2	
	S_1	S_1	S_1	S_1	S_1	
	S_2	S_2	S_2	S_2	S_2	

Рис. 1.3

Граничные сигналы:

$$\varphi_0^1 = \varphi_0^2 = 0.$$

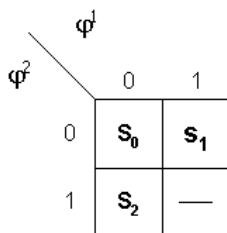
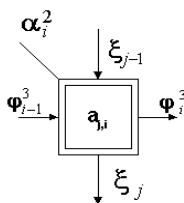


Рис. 1.4

	$\alpha_i^1 a_{j,i}$	00	01	11	10
$(\varphi^1 \varphi^2)_{i-1}$	00	00	10	00	01
	01	01	01	01	01
	11	11	11	11	11
	10	10	10	10	10

Рис.1.5

Поиск минимума выполняется по следующему алгоритму. Как и прежде, поиск ведется со старших разрядов. При анализе i -го бита j -я строка оставляется во множестве минимальных чисел, если $a_{j,i} = 0$. В случае $a_{j,i} = 1$ эта строка исключается из рассмотрения только тогда, когда для выделенного на $(i-1)$ -й итерации множества не все $a_{k,i}$ равны 1 ($1 \leq k \leq n$).



Входы элемента (рис. 1.6) таковы. Сигнал $\varphi_{i-1}^3 = 1$, если левая часть числа принадлежит множеству минимальных левых частей массива. Иначе $\varphi_{i-1}^3 = 0$. Сигнал $\xi_{j-1} = 1$, если на подмножестве минимальных левых частей сверху от элемента (j,i) все $a_{k,i} = 1$ ($k < j$). Иначе $\xi_{j-1} = 0$. Сигнал $\alpha_i^2 = \xi_n$.

Рис. 1.6.

Модифицированные таблицы переходов для правого и нижнего выходов (рис. 1.7, 1.8) дают:

$$\varphi_i^3 = \overline{a_{j,i}} \varphi_{i-1}^3 \vee \alpha_i^2 \varphi_{i-1}^3; \xi_j = a_{j,i} \xi_{j-1} \vee \xi_{j-1} \overline{\varphi_{i-1}^3}$$

Граничные сигналы:

$$\varphi_0^3 = \xi_0 = 1.$$

$$\alpha_i^2 \varphi_{i-1}^3$$

	$a_{j,i}$	0	1	φ_i^3
00		0	0	
01		1	0	
11		1	1	
10		0	0	

Рис. 1.7

$$\xi_{j-1} \varphi_{j-1}^3$$

	$a_{j,i}$	0	1	ξ_j
00		0	0	
01		0	0	
11		0	1	
10		1	1	

Рис. 1.8

Объединяя найденные решения, получаем схему элемента искомой среды (рис. 1.9). Значение выхода

$$\varphi_i^3 = (\overline{a_{j,i}} \vee \alpha_i^2) \varphi_{i-1}^3 \beta_j^2.$$

Сигнал $\beta_j^2 = \varphi_m^1$, если ведется поиск ближайшего большего, и $\beta_j^2 = 1$ при поиске минимума. В обоих этих случаях $\alpha_i^2 = \xi_n$. Для других видов процедур значения β_j^2 и α_i^2 безразличны.

По сигналу разрешения записи $\beta_j^1 = 1$ число $\alpha^1 = (\alpha_1^1 \dots \alpha_m^1)$ заносится поразрядно в j-ю строку. Считывание информации из l-й строки реализуется такой настройкой:

$$\xi_{o,j} = 1, \varphi_{1,0}^3 = 1, \varphi_{j,0}^3 = 0 \quad (j \neq 1).$$

При этом $\xi_{n,i} = a_{1,i}$. Так что специальный сигнал разрешения считывания не требуется.

Матрица $n_1 \times m_1$ из таких элементов имеет $M = 8n_1 + 4m_1$ внешних выводов. Дополнительно необходимо учесть два вывода подключения шин "земля" и "питание". Поэтому в корпусе БИС на 40 выводов размещается до 10 элементов синтезированной среды (матрица 2×5).

Практика 1. Моделирование 1

Для начала моделирования необходимо запустить файл lzs.exe. Моделирование состоит из демонстрационной и основной частей.

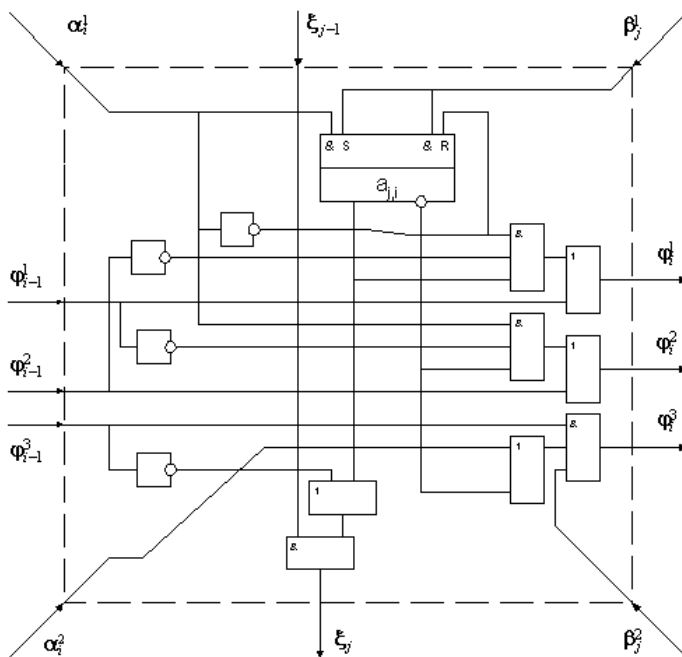


Рис. 1.9

В демонстрационной части показывается процесс реализации поиска на уровне элементов ЛЗС с иллюстрацией прохождения сигналов во времени через элементы среды. Входы и выходы элементов окрашиваются в синий или красный цвет, что соответствует логическим 0 или 1. Черный цвет выхода элемента означает, что данный выход еще не принял устойчивое состояние. После того, как все входы и выходы элемента определены, он изменяет свой цвет.

В основной (тестовой) части моделирование работы структуры происходит без показа протекающих в ней процессов. Программа тестирует моделируемую структуру на правильность функционирования, давая возможность студенту подтвердить на практике усвоенный ранее теоретический материал. При необходимости в процессе работы может быть вызвана нужная справка.

Общий вид окна после запуска программы представлен на рис. 1.10. С помощью меню или кнопок в панели управления можно

выбрать режим тестирования или запустить демо-версию программы. Предусмотрен вызов справки и выход из программы.

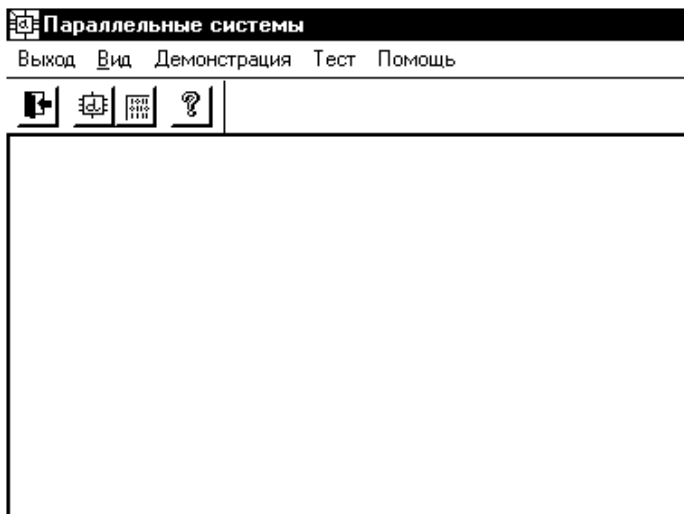


Рис. 1.10

Демонстрационная часть Общий вид окна после запуска демонстрационной программы представлен на рис.1.11, где изображено поле элементов среды размером (4×8), окрашенных первоначально в черный цвет (до запуска демонстрации). Внутри каждого элемента цифрой (0 или 1) обозначено его состояние. Ниже располагается ряд кнопок управления демонстрацией.

Для запуска демонстрации необходимо нажать одну из кнопок: “Поиск минимума” или “Поиск ближайшего большего”. Во время демонстрации происходит изменение цвета входов и выходов элементов, а также цвета самого элемента. Условное обозначение проводников при протекании по ним тока показано в левом верхнем углу. В верхней части окна отображается текущее состояние показа демонстрации.

Над кнопками расположены разряды признака α (слева направо). В нижней части окна располагается ряд кнопок, с помощью которых осуществляется управление программой:

Инфо

Вызов справки

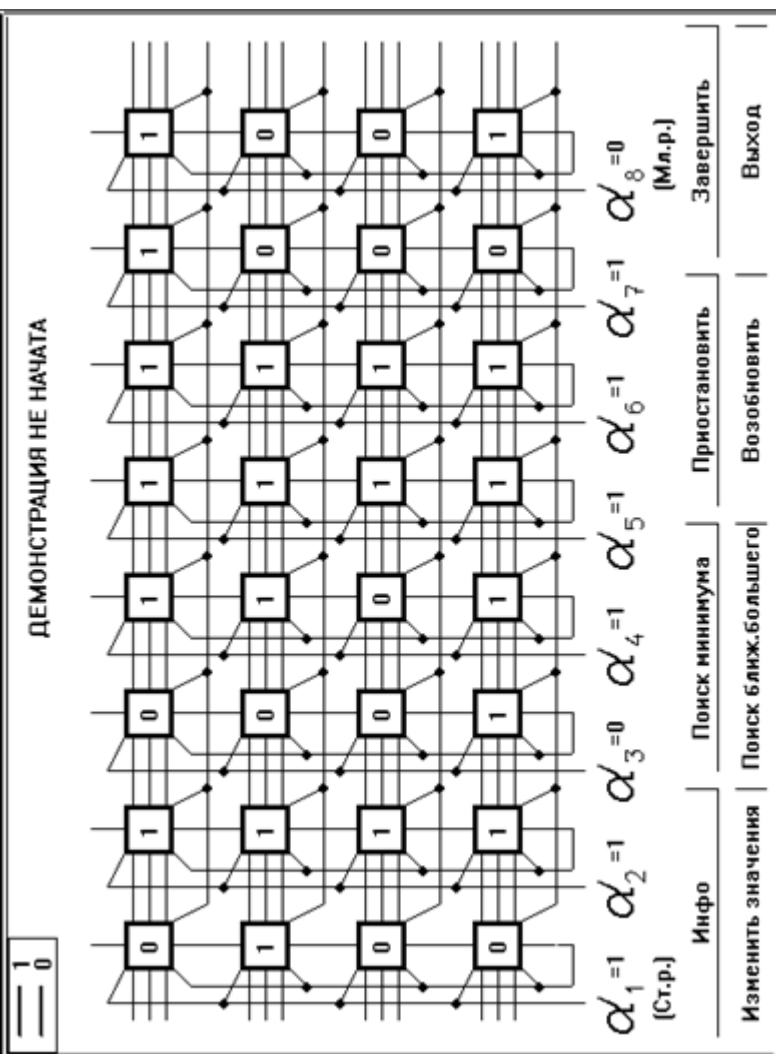


Рис. 1.11

Изменить значения	Изменить состояния элементов среды
Поиск минимума	Запуск демонстрации поиска минимума
Поиск ближ. большего	Запуск демонстрации поиска ближайшего большего
Приостановить	Приостановка демонстрации на время
Возобновить	Возобновление демонстрации после приостановки
Завершить	Закончить показ демонстрации
Выход	Выход из программы

Требуется, выбирая поиск минимума или поиск ближайшего большего для различных состояний элементов среды, проверить правильность выполнения и уяснить принцип работы данной ЛЗС.

Основная (тестовая) часть. Для запуска процесса тестирования надо выбрать в меню (рис. 1.10) пункт “Тест” или нажать соответствующую кнопку в панели управления. Появится запрос на размеры тестируемой матрицы (рис.1.12), где n - число строк матрицы (от 1 до 32), m - число столбцов матрицы (от 1 до 64).

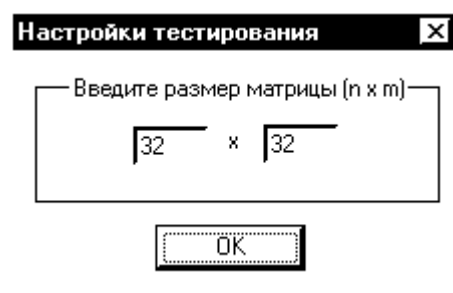


Рис. 1.12

В случае ввода недопустимого значения m или n будет выведено соответствующее предупреждение.

Далее появится запрос о способе ввода данных в матрицу (рис. 1.13).

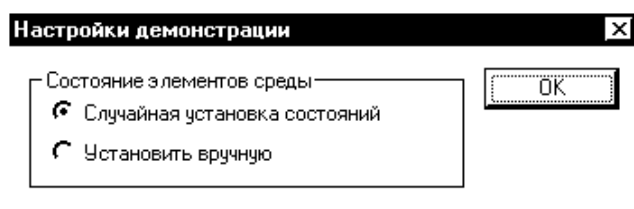


Рис. 1.13

Состояние элементов матрицы будет выбираться случайным образом или вводиться пользователем вручную.

Затем появится окно для ввода признака (рис. 1.14).

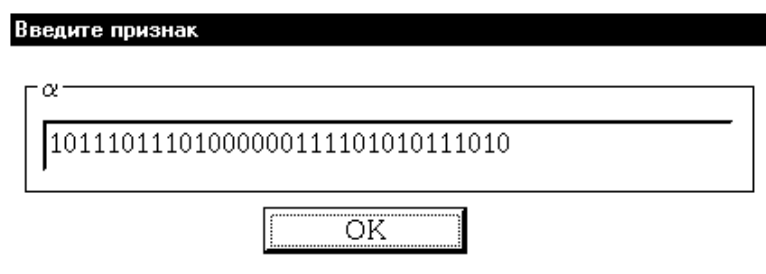


Рис. 1.14

Длина вектора-признака не должна превышать заданного количества столбцов. Ввод осуществляется со старших разрядов. Если вводится вектор меньшей длины, то недостающие младшие разряды заполняются нулями автоматически. Если введенное число окажется не двоичным, то будет дано соответствующее предупреждение.

Наконец, появится окно установки граничных сигналов (рис. 1.15). От правильности их установки зависит достоверность результатов тестирования.

Если был выбран режим ручного ввода данных, то на экране появятся границы матрицы в соответствии с введенными значениями n и m и курсор, показывающий текущее место ввода данных.

Ввод данных осуществляется нажатием клавиш "0" или "1" (рис. 1.16).

Режим ввода данных можно прервать, нажав клавишу "Esc". Нажатие любых других клавиш игнорируется и выдается предупреждающий звуковой сигнал.



Рис. 1.15

Если был выбран режим случайного ввода данных, то они генерируются случайным образом и выдаются на экран.

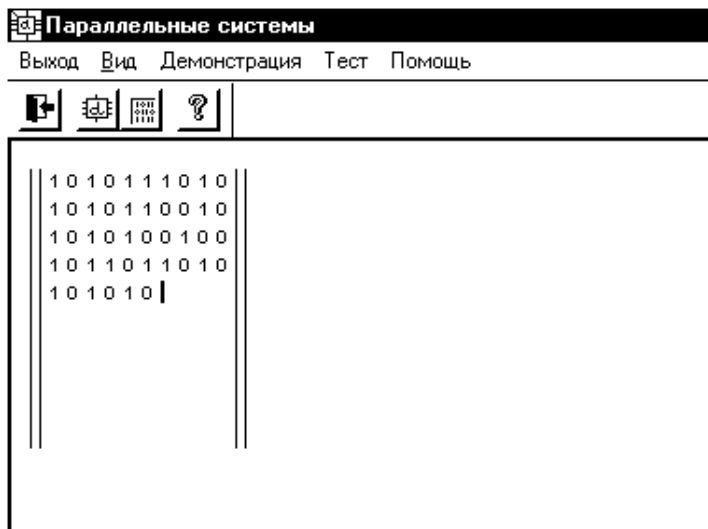


Рис. 1.16

По окончании ввода запускается алгоритм поиска минимума и ближайшего большего по отношению к признаку. Результаты тестирования выводятся на экран (рис.1.17).

Строка, выделенная зеленым цветом, отвечает ближайшему большему числу массива по отношению к признаку. Строка, выделенная красным цветом, определяет минимальное значение среди всех строк матрицы.

1	1	1	0	1	1	1	0	0	0	1	0	0	0	1
0	1	1	0	1	0	1	0	0	0	0	0	0	0	1
1	1	0	0	0	0	1	1	0	0	0	0	1	0	0
1	1	0	1	1	1	0	1	0	1	1	0	1	1	1
0	0	1	1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	1	0	0	1	0	1	1	0	0	1	0
0	0	1	0	1	1	0	1	1	1	1	1	1	1	0
0	0	1	0	1	1	1	1	0	0	1	0	0	0	0
1	1	0	1	0	0	1	1	0	0	1	1	1	0	1
1	0	0	0	1	0	0	0	1	1	1	0	0	1	1
0	1	0	1	1	1	1	1	0	1	1	1	0	1	0
1	0	0	0	1	0	1	1	0	0	0	0	1	0	1
1	0	1	1	0	0	1	1	0	0	1	0	1	1	0
0	0	0	1	1	0	1	1	0	1	0	1	1	1	0
0	1	1	1	1	1	1	0	1	1	1	0	1	0	1

Признак:
011101000000111

Рис. 1.17

В случае, если граничные сигналы были выставлены неправильно, выдается сообщение об ошибке (рис. 1.18).

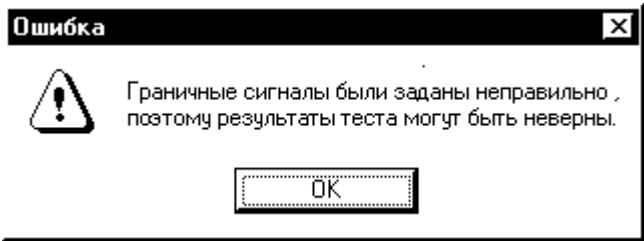


Рис. 1.18

Требуется несколько раз запустить процесс тестирования, задавая всякий раз граничные сигналы, и проверить правильность выполнения теста.

Занятие 2

СРЕДА ПОИСКА МИНИМАЛЬНОГО ПОКРЫТИЯ

Аппаратное решение задачи поиска минимального покрытия может быть использовано не только для ускорения процесса минимизации булевых функций, но и при поиске минимального набора тестов для обнаружения неисправностей логических схем и диагностических тестов. Аналогичные задачи возникают при оптимальном кодировании микроопераций, при минимизации и кодировании автоматных таблиц.

Лекция 2. Организация операционной матрицы [1,2]

Исходной информацией является импликантная матрица некоторой булевой функции. Строки матрицы отмечаются элементами множества импликант для этой функции. Столбцам соответствуют элементы множества M^1 наборов переменных, на которых значение функции равно 1. На пересечении строки и столбца ставится единица, если данный импликант покрывает соответствующий набор. Задача состоит в определении минимального подмножества импликант, которые в совокупности покрывают M^1 .

Приблизительно эту задачу можно решить следующим образом. В исходной матрице находится столбец с минимальным числом единиц. Если таких столбцов несколько, берется любой. Из строк, которые отмечены единицами найденного столбца, выделяется строка с максимальным числом единиц. Если таких строк несколько, берется любая. Соответствующий ей импликант вводится в решение. Столбцы, отмеченные единицами этой строки, исключаются из дальнейшего рассмотрения. Процесс повторяется до тех пор, пока не будут исключены все столбцы.

Алгоритм. Предусмотрим в исходной матрице X_a маскирование разрядов по горизонтали (маска $MX = \|m_i\|$) и вертикали (маска $MY = \|m_j\|$). Тогда применительно к использованию ЛЗС возможна такая формулировка алгоритма поиска минимального покрытия.

1. $MX := \|1\|$ (единичная матрица-строка).

2. Поиск столбца с минимальным количеством единиц. Если результат неоднозначен, берется первый слева. Пусть это будет столбец X_a^i .

3. Формирование вертикальной маски, $MY := X_a^i$.

4. Поиск строки с максимальным числом единиц. Если результат неоднозначен, берется первая сверху. Пусть это будет строка X_a^j .

5. Код (отмечающая импликанта) строки X_a^j вводится в решение.

6. Формирование горизонтальной маски, $MX := MX \& \overline{X_a^j}$ (операции выполняются поразрядно).

7. Если $MX \neq ||0||$, идти к п.2. Иначе КОНЕЦ.

Реализация всех пунктов алгоритма, исключая п.2 и 4, вполне очевидна. Для отыскания столбца с минимальным числом единиц будем зачеркивать в каждом такте по одной единице во всех незамаскированных столбцах. Такому подходу отвечает элемент, показанный на рис. 1.19.

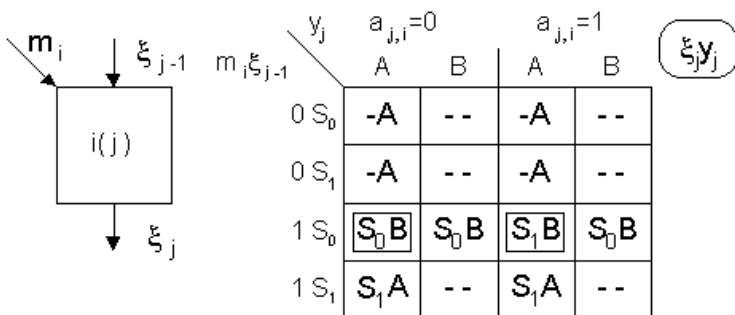


Рис. 1.19

Для однозначной индикации результата в каждом столбце применяем дополнительный $(n+1)$ -й элемент с безразличным значением $a_{n+1,i}$. В искомом столбце этот элемент "вычеркнут" (пе-

реведен в состояние В) раньше, чем в других столбцах. Одновременно прекращается поступление синхроимпульсов. Строка дополнительных элементов образует обрамление среды снизу (рис. 1.20).

Для реализации приоритетного опроса этой строки слева сигналом T_0 элементу $(n+1,i)$ придается комбинационная часть (рис. 1.21).

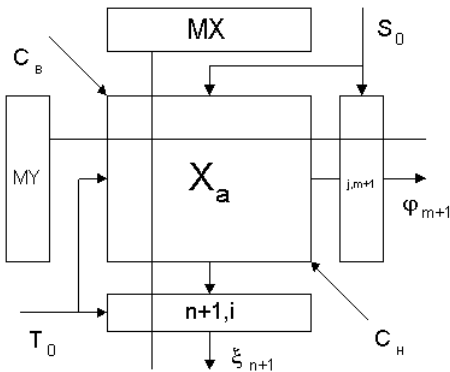


Рис. 1.20

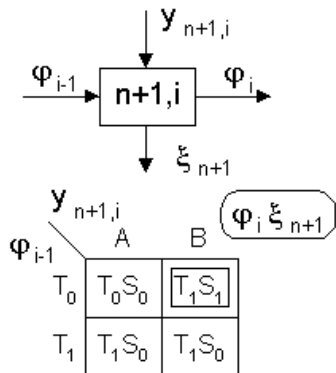


Рис. 1.21

Сигнал $\xi_{n+1} = S_1$ появляется на выходе первого слева "вычеркнутого" дополнительного элемента.

Поиск строки с максимальным числом единиц выполняется аналогично. Но теперь в каждом такте вычеркиваем по одному нулю во всех незамаскированных строках (рис. 1.22).

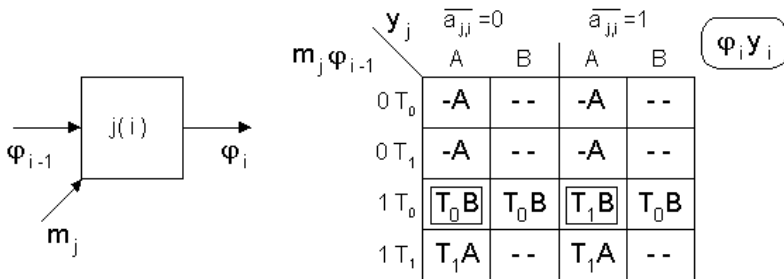


Рис. 1.22

Поступление синхроимпульсов прекращается, как только дополнительный (m+1)-й элемент хотя бы одной из строк "вычеркнут". Столбец дополнительных элементов образует обрамление среды справа (рис. 1.20).

Приоритетный опрос этого столбца сверху сигналом S_0 реализует специальная комбинационная приставка (рис. 1.23) к элементу (j,m+1). Выделяется строка, на выходе которой сигнал $\varphi_{m+1} = T_1$.

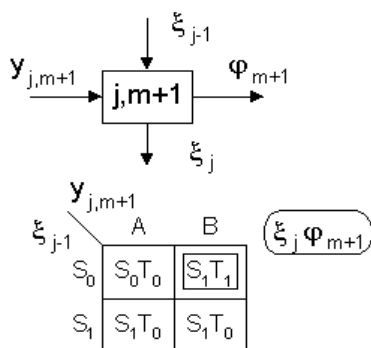


Рис. 1.23

Сигналы S_0 и T_0 по краям среды действуют постоянно. Первоначально все элементы установлены в состояние A. Матрица X_a содержит два слоя элементов ("верхний" и "нижний") со своими синхросигналами (\tilde{N}_a и \tilde{N}_i). Каждый элемент "верхнего слоя" отвечает таблице на рис. 1.19. Пусть $A = S_0 = T_0 = 0$ и $B = S_1 = T_1 = 1$.

Из кодированной таблицы (рис.1.24) имеем:

$$\xi_j = \xi_{j-1} \vee a_{j,i} \overline{y_j} ; y_j = m_i \overline{\xi_{j-1}}.$$

$m_j \xi_{j-1} \backslash a_{j,i} y_j$					
		00	01	11	10
00	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>10</u>
01	<u>10</u>	<u>10</u>	<u>10</u>	<u>10</u>	<u>10</u>
11	<u>10</u>	<u>10</u>	<u>10</u>	<u>10</u>	<u>10</u>
10	<u>01</u>	<u>01</u>	<u>01</u>	<u>01</u>	<u>11</u>

Рис. 1.24

Соответственно получаем схему "верхнего" элемента (рис. 1.25).

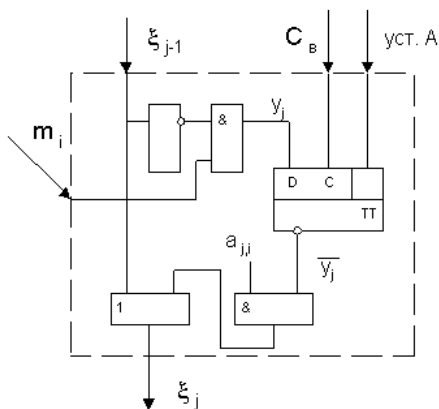


Рис. 1.25

Схема элемента "нижнего слоя" аналогична (с переобозначением сигналов и подачей на вход сигнала $\overline{a_{j,i}}$).

Наличие D-триггера в дополнительном элементе (n+1,i) не обязательно, поскольку правильное значение $y_{n+1,i} = m_i \overline{\xi_n}$ получается сразу по вычеркивании нижней единицы в столбце (наличие последующих невычеркнутых нулей оставляет значение $\xi_j = \xi_{j-1} = 0$). С учетом таблицы рис. 1.21 имеем схему (рис. 1.26),

где

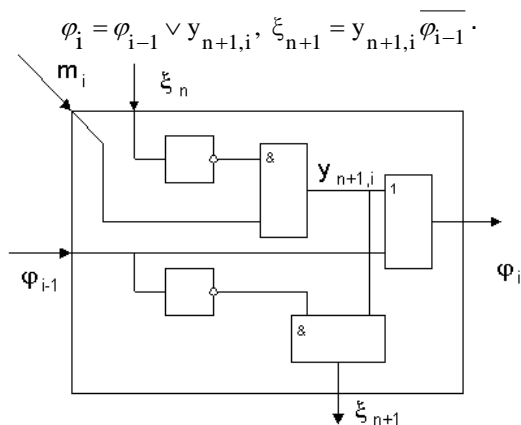


Рис. 1.26

Элемент (j,m+1) аналогичен.

Практика 2. Моделирование 2

Для начала моделирования необходимо запустить файл lzs2.exe. Моделирование состоит из демонстрационной и основной частей:

В демонстрационной части показывается процесс реализации поиска на уровне элементов ЛЗС с иллюстрацией прохождения сигналов во времени через элементы среды.

В основной (тестовой) части моделирование работы структуры происходит без показа протекающих в ней процессов. Программа тестирует моделируемую структуру на правильность функционирования, давая возможность студенту подтвердить на практике усвоенный ранее теоретический материал. При необходимости в процессе работы может быть вызвана нужная справка.

Общий вид окна после запуска программы представлен на рис. 1.27. С помощью меню или кнопок в панели управления можно выбрать режим тестирования или запустить демо-версию программы. Предусмотрен вызов справки и выход из программы.

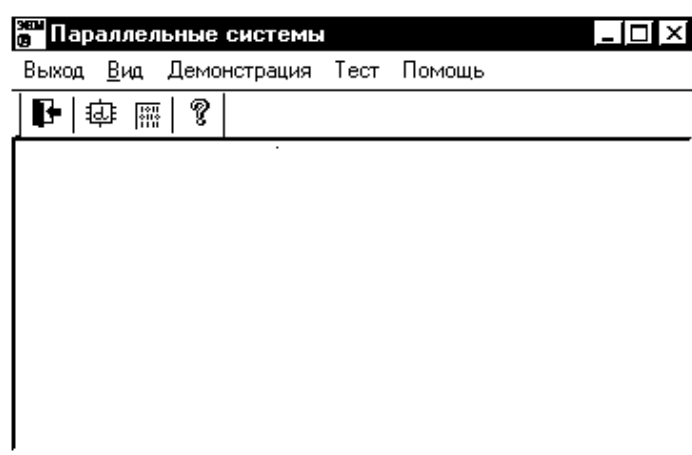





Рис. 1.27

Имеются следующие пункты меню (и соответствующие им кнопки в панели управления):



Выход

Выход из программы

	Демонстрация	Запуск демонстрационной модели ЛЗС
	Тест	Формирование и тестирование модели ЛЗС, созданной студентом
	Помощь	Вывод на экран справки

Для запуска демонстрационной модели необходимо выбрать в меню пункт “Демонстрация” или нажать соответствующую кнопку в панели управления. После нажатия на экране появится окно демонстрации (рис. 1.28). После ознакомления с демонстрационной моделью нужно перейти к работе с тестовой версией программы.

Демонстрационная часть. Общий вид окна после запуска демонстрационной программы представлен на рис. 1.28. В его центре условно обозначено поле логических элементов ЛЗС размером (5×7). Основные элементы среды схемно представлены парой квадратов, связи – линиями, разряды маскирующих векторов – окружностями. Места соединений линий связи и выводов обозначены узлами, как на обычных чертежах. Внутренние состояния триггеров каждого элемента обозначаются буквенной информацией (“А” – соответствует логическому ‘0’, “В” – соответствует логической ‘1’), а логические значения выводов – графической (логическим значениям сопоставлены определенные цвета). Дополнительные элементы обозначены квадратами с диагональной сеткой внутри.

Во время демонстрации происходят изменения цветов линий, соединяющих элементы, что означает соответствующие изменения логических уровней протекающих по ним токов. Черный цвет выхода элемента означает, что данный выход еще не принял устойчивого состояния. Два других возможных цвета (красный и синий) устанавливаются через определенное время (моделируется задержка формирования выходов внутри элемента) после того, как соответствующие входы примут стационарные значения. В случае, когда с приходом тактового импульса изменяется внутреннее состояние основного элемента, выход окрашивается в черный цвет. Это происходит в связи с тем, что для основного элемента выход является функцией входов и состояния. Синий цвет выхода соответствует логическому “0”, красный – логической “1”.

В верхней части окна отображается текущее состояние показа демонстрации. В нижней части окна располагается ряд кнопок, с помощью которых осуществляется управление программой:

Параллельные системы. Лабораторная работа №2

ДЕМОНСТРАЦИЯ НЕ НАЧАТА

1
0

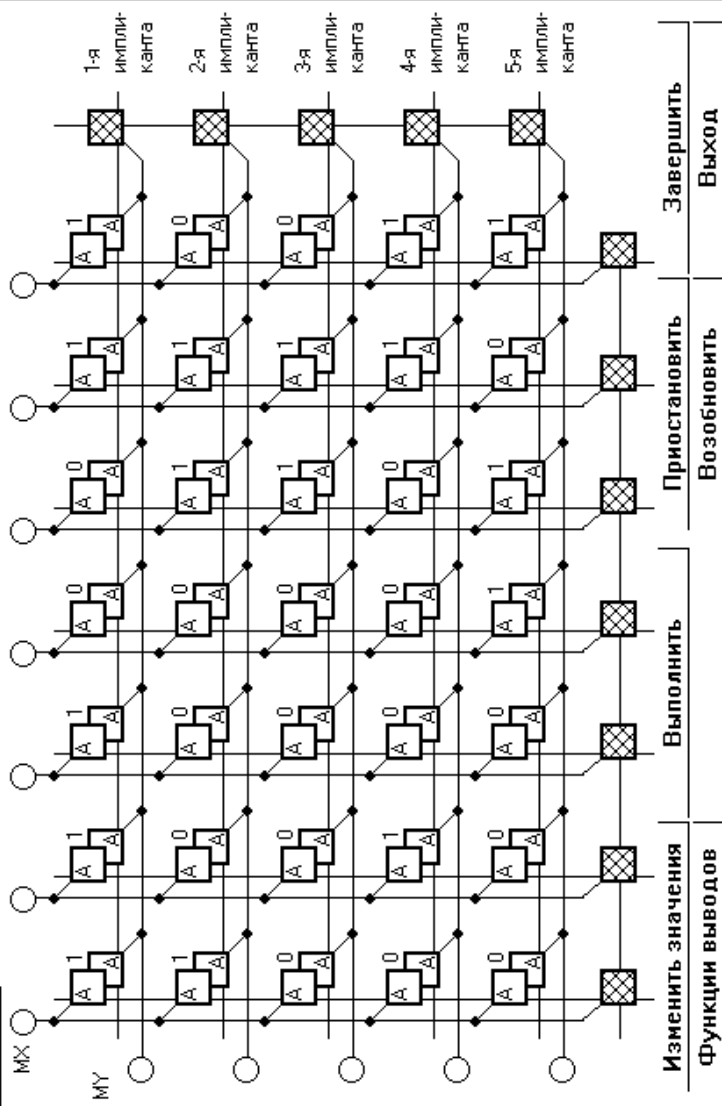


Рис. 1.28

Функции выводов	Вызов справки
Изменить значения	Изменить состояния элементов среды
Выполнить	Запуск демонстрации
Приостановить	Приостановка демонстрации
Возобновить	Возобновление демонстрации после приостановки
Завершить	Закончить показ демонстрации
Выход	Выход из программы

Требуется, запуская демонстрацию для различных состояний элементов среды, проверить правильность выполнения и уяснить принцип работы данной ЛЗС.

Основная (тестовая) часть. Для запуска процесса тестирования надо выбрать в меню рис. 1.27 пункт “Тест” или нажать соответствующую кнопку в панели управления.

Появится запрос на размеры m и n тестируемой матрицы (рис. 1.29), где n – число строк матрицы (от 1 до 32), m – число ее столбцов (от 1 до 64).

Рис. 1.29

В случае ввода недопустимого значения m или n будет выведено соответствующее предупреждение.

Далее появится запрос о способе ввода данных в матрицу (рис.1.30) –вручную или автоматически со случайным заполнением.

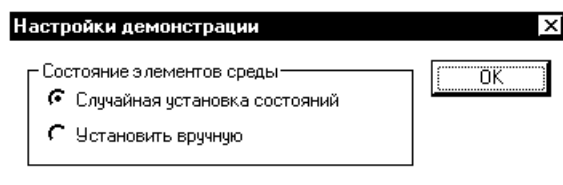


Рис. 1.30

Затем появится окно установки граничных сигналов (рис. 1.31). От правильности их установки зависит достоверность результатов тестирования.



Рис. 1.31

Если был выбран режим ручного ввода данных, то на экране появятся границы матрицы (в соответствии с введенными значениями n и m) и курсор, показывающий текущее место ввода данных (рис. 1.32). Ввод данных осуществляется нажатием клавиш "0" или "1". Режим ввода данных можно прервать, нажав клавишу "Esc". Нажатие любых других клавиш игнорируется.

Если был выбран режим случайного ввода данных, то они генерируются случайным образом и выдаются на экран. После этого

запускается алгоритм поиска минимального покрытия, и результаты тестирования выводятся на экран (рис. 1.33). Строки, выделенные прямоугольником, образуют минимальное покрытие.

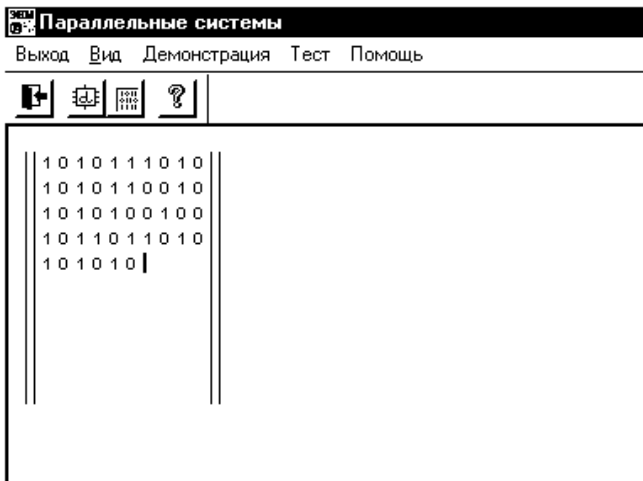


Рис. 1.32

В случае, если граничные сигналы были выставлены неправильно, выдается сообщение об ошибке.

0	1	0	0	0	1	0	1	1	0	0	0	0	0	0
0	1	1	0	1	1	1	1	1	0	1	0	1	0	1
1	1	1	0	1	0	0	0	1	1	0	1	0	1	1
1	1	0	0	0	0	1	1	0	1	0	1	0	1	0
1	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	1	1	1	1	0	1	1	0	1	0	0	1	1	1
1	0	0	1	0	0	1	0	0	1	1	0	1	1	0
1	0	1	1	0	0	0	1	0	0	1	1	1	0	1
0	1	1	1	0	1	1	1	1	0	0	1	0	0	0
1	0	1	0	0	1	0	0	0	0	1	1	0	0	
0	0	0	0	0	1	1	1	0	1	1	1	0	1	
1	0	0	0	1	0	1	1	0	1	1	0	1	0	
1	0	0	0	0	1	0	1	0	0	1	1	0	0	1
1	0	0	1	0	0	1	1	1	0	1	1	0	0	
1	1	0	0	0	1	1	1	1	0	1	1	0	0	

Рис. 1.33

Требуется несколько раз запустить процесс тестирования, задавая всякий раз граничные сигналы, и проверить правильность выполнения теста.

Занятие 3

СРЕДА ОДНОТАКТНОГО РАСПОЗНАВАНИЯ

Лекция 3. Матрица распознавания [3]

Базовая операционная матрица A представляет собой специализированную логику-запоминающую среду (рис. 1.34, а), каждый элемент (j,i) которой содержит триггер для хранения элемента a_{ji} анализируемого двоичного массива (кадра) A ($j = 1 \dots k, i = 1 \dots \ell$) и дополнительную логику. По условию сигнал $z^{ji}=1$ на внешнем выходе элемента (j,i) появляется, если этот элемент отвечает правой нижней границе заданной троичной матрицы (эталона) X .

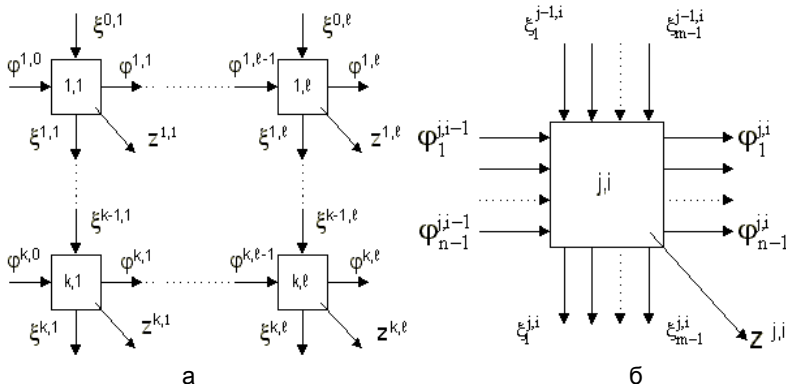


Рис. 1.34

При этом построение среды начинается с n -столбца матрицы A . Однако требуется k дополнительных $(n-1)$ -разрядных регистров для задания граничной матрицы $\|a_{jq}\|, j = 1 \dots k$. Боковые выходы элемента (j,i) совпадают с боковыми входами элемента $(j,i+1)$. Поэтому

$$\varphi_q^{ji} = \|\varphi_q^{ji}\|, \quad \varphi_q^{ji} = \varphi_{q+1}^{j,i-1}, \quad \varphi_n^{j,i-1} = a_{ji}.$$

Обозначим $X_r = \|x_{pq}\|, p = 1 \dots r, r = 1 \dots (m-1)$. Будем говорить, что элемент (j,i) является r -границей образа X , если этот элемент отвечает правой нижней границе матрицы X_r . Введем множество вертикальных входов элемента (рис. 3.1б), так чтобы оно давало

для любого допустимого значения $r \in \{1 \dots (m-1)\}$ ответ на вопрос, является ли предыдущий верхний элемент r -границей образа X :

$$\xi_{r-1,i} = \|\xi_r^{j-1,i}\|, \quad \xi_r^{j-1,i} = \bigwedge_{s=1}^r \bigwedge_{q=1}^n (\varphi_q^{j-(r-s+1),i-1})^{x_{sq}},$$

$$(\varphi_q^{j-1,i})^{x_{sq}} = \begin{cases} \varphi_q^{j-1,i}, & x_{sq}=1; \\ \neg \varphi_q^{j-1,i}, & x_{sq}=0; \\ 1, & x_{sq} \in \{-\}. \end{cases}$$

Соответственно вертикальные выходы

$$\xi_r^{ji} = \xi_{r-1}^{j-1,i} \bigwedge_{q=1}^n (\varphi_q^{j,i-1})^{x_{rq}}, r=1 \dots (m-1).$$

Значение $\xi_0^{j-1,i} = 1$. Граничные сигналы $\xi_r^{0,1} = 0$. Внешний выход

$$z^{ji} = \xi_m^{ji} = \xi_{m-1}^{j-1,i} \bigwedge_{q=1}^n (\varphi_q^{j,i-1})^{x_{mq}}.$$

По условию структура должна допускать последовательные опросы по множеству эталонов X неизменных размеров, что возможно только при использовании перестраиваемых модулей. В связи с этим представим выражение для $\xi_r^{j,i}$, $r = 1 \dots m$, в виде:

$$\xi_r^{ji} = \xi_{r-1}^{j-1,i} \bigwedge_{q=1}^n (\varphi_q^{j,i-1} x_{rq} \vee \neg \varphi_q^{j,i-1} \neg x_{rq} \vee m_{rq}).$$

Здесь $\|x_{rq}\|$ – матрица X , в которой безразличные состояния заполнены произвольным образом, и m_{rq} – элемент матрицы масок $\|m_{rq}\|$ тех же размеров, что и матрица X :

$$m_{rq} = \begin{cases} 0, & x_{rq} \in \{0,1\}; \\ 1, & x_{rq} \in \{-\}. \end{cases}$$

Отсюда следует возможность реализации элемента на основе ассоциативного модуля (рис. 1.35) с выходом z типа "открытый коллектор", предназначенного для анализа строчных фрагментов разрядностью $q \leq n$. В модуле используется специальная схема RS-триггера с дополнительным управляющим входом V . Нормально

$V=1$. Если же вместо триггера требуется образовать последовательный каскад из двух инверторов, что необходимо для стыковки модулей с целью наращивания разрядности при $n>q$, то $V=0$.

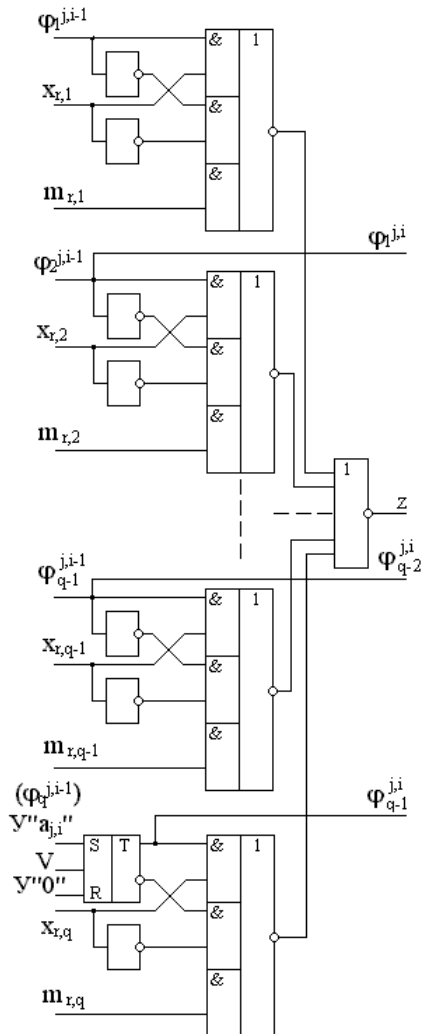


Рис. 1.35

В табл. 1.1 приведены вычисленные параметры структуры для $\ell = 64$, когда $q=q_{\text{опт}}$. Здесь $q_{\text{опт}}$ – значение q , минимизирующее

W при данных $N_{\text{БИС}}$, n и ℓ ; W – число корпусов; Q_M – горизонтальный размер фрагмента структуры, размещаемого в модуле БИС с числом сигнальных выводов $N_{\text{БИС}}$.

Таблица 1.1

$N_{\text{БИС}}$	n	12	10	8	6	4
60	$q_{\text{опт}}$	7	10	8	6	4
	Q_M	11	7	9	12	15
	$K_{\text{БИС}}$	330	294	306	312	270
	$W_{\text{мин}}/\text{km}$	10	8	7	5	5
46	$q_{\text{опт}}$	7	6	8	6	4
	Q_M	6	7	5	7	10
	$K_{\text{БИС}}$	180	182	170	182	180
	$W_{\text{мин}}/\text{km}$	18	16	12	9	7

Как следует из табл.1.1, реализация матриц значительных размеров для идентификации сравнительно крупных объектов при современном уровне технологии практически затруднена. Так, уже в случае $k = l = 64$ и $n = m = 12$ требуется 7680 корпусов БИС на 60 выводов. С уменьшением количества выводов необходимое количество корпусов быстро нарастает.

Практика 3. Моделирование 3

Для начала моделирования необходимо запустить файл `lzs3.exe`. Моделирование состоит из демонстрационной и основной частей.

В демонстрационной части показывается процесс распознавания на уровне элементов ЛЗС с иллюстрацией прохождения сигналов во времени через элементы среды.

Входы и выходы элементов окрашиваются в синий или красный цвет, что соответствует логическим 0 или 1. Черный цвет выхода элемента означает, что данный выход еще не принял устойчивое состояние. После того, как все входы и выходы элемента определены, он изменяет свой цвет.

В основной (тестовой) части моделируется работа структуры без показа самого процесса. Программа тестирует моделируемую структуру на правильность функционирования, давая возможность студенту подтвердить на практике усвоенный ранее теоретический

материал. При необходимости в процессе работы может быть вызвана нужная справка.

Общий вид окна после запуска программы представлен на рис. 1.36. С помощью меню или кнопок в панели управления можно выбрать режим тестирования или запустить демо-версию программы. Также предусмотрен вызов справки и выход из программы.

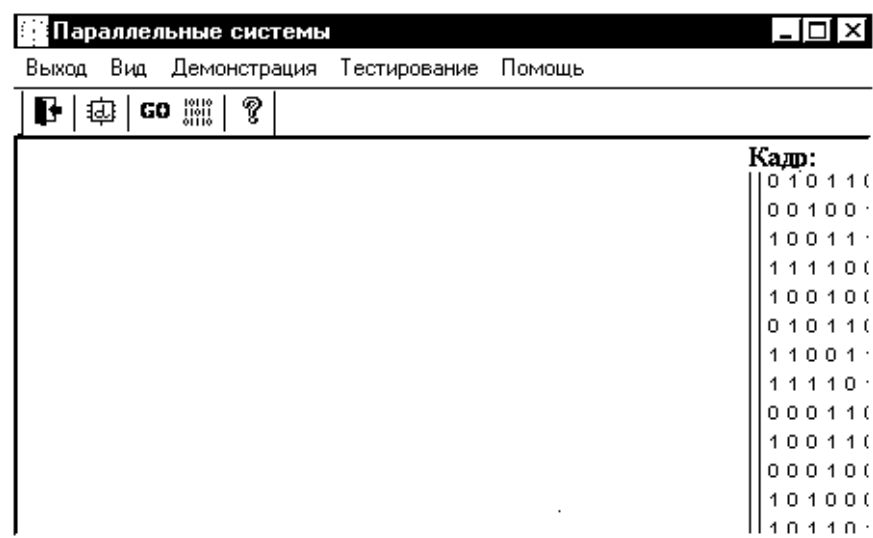


Рис. 1.36

Имеются следующие пункты меню (и соответствующие им кнопки в панели управления):



Выход

Выход из программы



Демонстрация

Запуск демонстрационной модели ЛЗС



Размер двоичного массива

Ввод размеров двоичного массива



Тестирование

Формирование и тестирование модели ЛЗС, созданной студентом



Помощь

Вывод на экран справки

Для запуска демонстрационной модели необходимо выбрать в меню пункт “Демонстрация” или нажать соответствующую кнопку в панели управления. После нажатия на экране появится окно программы демонстрации (рис. 1.37). После ознакомления с демонстрационной моделью необходимо перейти к работе с тестовой версией программы.

Демонстрационная часть. Демонстрационная часть моделирует процесс распознавания эталона размером 3×3 в двоичной матрице размером 25×25 элементов. Пользователь может посмотреть процесс изменения состояния элементов среды в окне 3×3 .

Общий вид окна после запуска демонстрационной программы представлен на рис. 1.37. На рисунке изображено два поля элементов среды: поле двоичной матрицы размером (25×25) , поле просмотра размером (3×3) . Поле матрицы просмотра представлено набором элементов ЛЗС. Внутри каждого элемента цифрой (0 или 1) обозначено его состояние. Ниже располагается ряд кнопок управления демонстрацией.

Для запуска демонстрации необходимо предварительно в поле двоичной матрицы установить с помощью мыши окно просмотра и нажать кнопку “Выполнить”. По желанию пользователя можно предварительно изменить состояния матрицы-эталона и двоичной матрицы. Во время демонстрации происходит изменение цветов входов – выходов элементов и самих элементов.

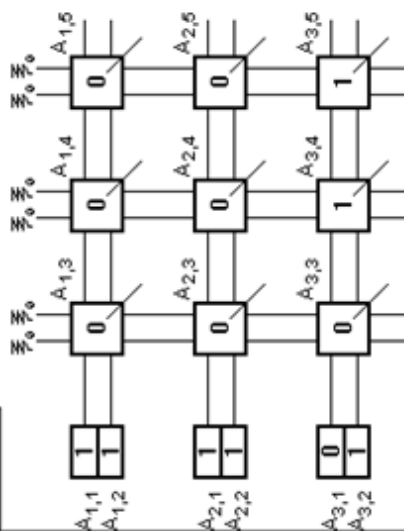
Функциональные назначения кнопок:

Приостановить	Приостановка демонстрации
Возобновить	Возобновление демонстрации после приостановки
Выполнить	Запуск демонстрации
Сменить эталон	Смена состояний эталона
Сменить матрицу A	Смена состояний двоичной матрицы
Завершить	Закончить показ демонстрации
Выход	Выход из программы

В ходе демонстрации важно понимать, что процесс распознавания идет по всей двоичной матрице (25×25) , но процесс поиска наблюдается только в выбранном окне (3×3) .

10

ДЕМОНСТРАЦИЯ НЕ НАЧАТА



НАЙДЕНО(строка,столбец):

Выберите область нажатием
левой клавиши мыши

Выполнить

Приостановить

Завершить

Сменить эталон

Изменить матрицу A|

Инфо

Возобновить

Вывод

Рис. 1.37

Основная (тестовая) часть. Для запуска процесса тестирования надо выбрать в меню (рис. 1.36) пункт “Тестирование” или нажать соответствующую кнопку в панели управления.

Появится запрос на размеры матрицы-эталона (рис. 1.38), где n – число строк матрицы (от 1 до 32), m – число столбцов матрицы (от 1 до 32).

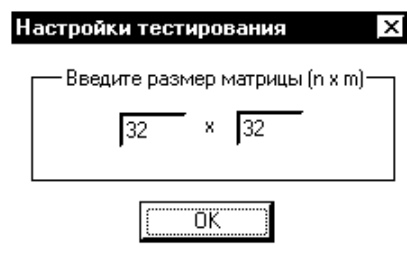


Рис. 1.38

В случае ввода недопустимого значения m или n будет выведено соответствующее предупреждение.

Далее появится запрос о способе ввода данных в матрицу (рис. 1.39).

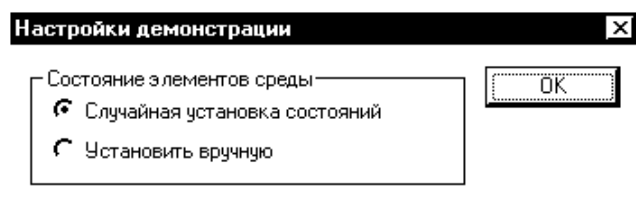


Рис. 1.39

Состояния элементов матрицы будут выбираться случайным образом или вводиться пользователем вручную.

Если данные вводятся вручную, то режим ввода можно прервать, нажав клавишу “Esc”. Нажатие любых других клавиш игнорируется. При этом выдается предупреждающий звуковой сигнал.

Если был выбран режим случайного ввода данных, то они генерируются случайным образом и выдаются на экран.

По окончании ввода появится окно результатов (рис. 1.40). В этом окне выводится список координат всех найденных вхождений матрицы-эталона в двоичную матрицу.

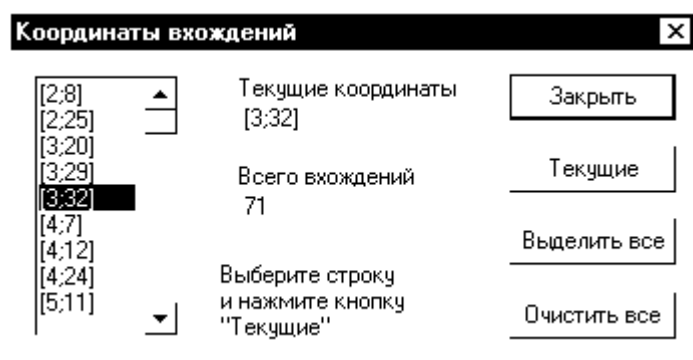


Рис 1.40

Кнопки "Текущие" и "Выделить все" позволяют просмотреть все найденные вхождения сразу или по одному. При нажатии на кнопку "Текущие" в двоичной матрице-кадре выделяется то вхождение, координаты которого отмечены курсором. Выделение текущих координат производится щелчком курсора мыши на нужную строку и нажатием кнопки "Текущие".

Занятие 4

МНОГОТАКТНОЕ РАСПОЗНАВАНИЕ. МАТРИЧНЫЙ СПЕЦПРОЦЕССОР-ИДЕНТИФИКАТОР

Лекция 4. Многотактный алгоритм и его реализация [3]

Необходимость многотактного распознавания. Рассматриваемую задачу идентификации сформулируем следующим образом. Для данного троичного эталона (троичной матрицы)

$$X^t = \|x_{pq}^t\|; x_{pq}^t \in \{0,1,-\}; p = 1 \dots m^t; q = 1 \dots n^t; t = 1 \dots \gamma$$

(t – номер типа объекта, γ – число типов объектов) найти координаты всех покрываемых им объектов двоичного изображения (булевой матрицы)

$$A = \|a_{ji}\|; a_{ji} \in \{0,1\}; j = 1 \dots K; i = 1 \dots L; (K \geq m^t, L \geq n^t).$$

Однотактное аппаратное решение этой задачи (см. занятие 3) предполагает использование определенным образом организованной матрицы операционной логико-запоминающей среды на основе ассоциативных модулей БИС специального вида. Однако реализация однотактных матриц практически затруднена из-за чрезмерного числа корпусов, которое быстро увеличивается с ростом размеров объектов и изображения в целом.

Выходом из затруднения является анализ изображения последовательно по кадрам и многотактно – фрагментами $(1 \times 1)n_1$ бит, $n_1 \geq 2$ – для каждого эталона. Тогда в кадре $k \times \ell$ бит оказывается возможным распознавание объектов различных размеров $(d \times c)n_1$, где $d = 1 \dots [k/n_1]$ и $c = 1 \dots [\ell / n_1]$, при разумной сложности операционной матрицы. Однако многотактная организация процедуры распознавания приводит к необходимости разработки спецпроцессора-идентификатора, работающего в комплексе с универсальной ЭВМ.

Эта ЭВМ выполняет управляющие функции: диспетчирование, прием-передача и формирование массивов данных для обработки и хранения. Спецпроцессор выполняет параллельную часть обработки. Основной вклад в объем его оборудования вносит обрабатывающая часть. Помимо операционной матрицы, она содержит многослойную буферную память с параллельным обменом данными между каждым слоем буфера и матрицей. В спецпроцессоре имеется свое устройство управления, оперативная память и так называемый координатный блок.

Существуют реальные возможности аппаратного анализа кадров размерами 128×128 . Процесс распознавания произвольных объектов может быть построен на матрицах со значением n и m от 2 до 4. При $n = m = 4$ количество корпусов $W = 2816$. Если же $n = m = 2$, то $W = 1024$. Это приемлемо.

Параллельный алгоритм. Задачей занятия является рассмотрение принципов организации спецпроцессора-идентификатора и сравнительная оценка скорости выполнения рассматриваемой процедуры идентификации на последовательной ЭВМ и комплексе: спецпроцессор – базовая ЭВМ. Решение первой части задачи будем связывать с разработкой алгоритма многотактного распознавания. Нужную сравнительную оценку получим путем детали-

зации (до уровня машинных команд) алгоритмов идентификации для последовательной ЭВМ и комплекса.

Размеры всех объектов будем полагать одинаковыми, а сами объекты – непересекающимися. Оценку дадим для случая "шашечного" расположения объектов (рис. 1.41,а), когда количество объектов на каждом "этаже" одинаково и равно ε . В качестве варьируемых параметров возьмем размеры объектов $m \times n$, изображения в целом $K \times L$, число типов объектов γ . Анализ проведем в предположении наличия между объектами непрерывной фоновой помехи, полагая величины L , ℓ и n кратными байту.

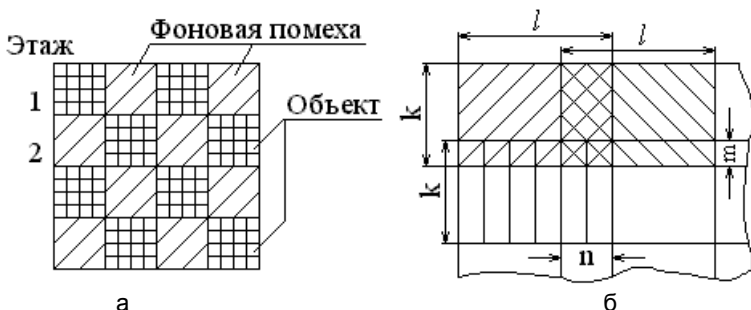


Рис. 1.41

Идентификацию на комплексе будем проводить в два этапа:

1. Формирование кадров – реализуется на базовой ЭВМ.
2. Идентификация объектов каждого кадра, последовательно по кадрам – реализуется на спецпроцессоре.

При работе с отдельным кадром $k \times \ell$ бит распознаются лишь те объекты, которые полностью входят в кадр. Поэтому кадры формируются с перекрытием (рис. 1.41,б): по горизонтали – на n бит, по вертикали – на m бит. Число кадров по горизонтали $Q = \lfloor (L - n) / (\ell - n) \rfloor$ и по вертикали $P = \lfloor (K - m) / (k - m) \rfloor$. Общее количество кадров в изображении равно PQ .

Принципы организации обрабатывающей части спецпроцессора можно уяснить из рассмотрения параллельного алгоритма многотактного распознавания объектов некоторого типа в анализируемом кадре. По условию каждый эталон последовательно и однозначно разделяется на cd непересекающихся фрагментов $\Lambda_{su} = \|\lambda_{pq}\|_{su}$ ($p, q = 1 \dots n_1$; $n_1 \geq 2$; $s = 1 \dots c$; $u = 1 \dots d$).

Алгоритм:

1. $w := 1, u := 1$.
2. $s := 1$.
3. Опросить кадр на Λ_{su} .
4. По результатам опроса сформировать двоичную матрицу $X' = ||x'_{ji}||$ ($j = n_1 \dots k; i = n_1 \dots \ell$).
5. $s := s + 1$.
6. Если $s \leq c$, идти к шагу 7. Иначе переход к шагу 10 (для $w=1$) или к шагу 11 (при $w=2$).
7. Опросить кадр на Λ_{su} .
8. По результатам опроса сформировать матрицу $X'' = ||x''_{ji}||$ и образовать ее покомпонентную дизъюнкцию с X' .
9. Опросить матрицу $(X' \vee X'')$ на признак-строку $\Gamma = ||1' - \dots - 1''||$, где число безразличных состояний равно $(n_1 - 1)$. Переход к шагу 4.
10. По результатам опроса сформировать матрицу $Y' = ||y'_{ji}||$. Переход к шагу 13.
11. По результатам опроса сформировать матрицу $Y'' = ||y''_{ji}||$ и образовать ее покомпонентную дизъюнкцию с Y' .
12. Опросить матрицу $(Y' \vee Y'')$ на признак-столбец ГТ (транспонированная матрица Г). Переход к шагу 10.
13. $w := 2; u := u + 1$.
14. Если $u \leq d$, идти к шагу 2. Иначе к шагу 15.
15. Результат опроса принять за конечный результат.

На рис. 1.42 дана сжатая последовательная иллюстрация процесса многотактного распознавания по предложенному алгоритму в случае $k = 6; \ell = 14; m = 4; n = 6; n_1 = 2$ ($c = 3, d = 2$) на примере матриц:

$$A = \left\| \begin{array}{cccccccccccccccc} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array} \right\|, \quad X = \left\| \begin{array}{ccc|cc} \Lambda_{11} & \Lambda_{21} & \Lambda_{31} & & \\ 1 & - & - & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & - \\ 1 & 1 & - & 1 & - & 1 \\ \Lambda_{12} & \Lambda_{22} & \Lambda_{32} & & \end{array} \right\|.$$

$$\omega = 1, u = 1$$

	2	4	6	8	10	12	14
2			1'		1'		1'
4	1'		1'		1'		1'
6		1'		1'		1'	

а

$$\omega = 2, u = 2$$

	2	4	6	8	10	12	14
2				1'' 1'			
4	1'		1'				
6	1'	1''	1'' 1'	1'' 1'	1'' 1'	1'' 1'	

е

	2	4	6	8	10	12	14
2			1' 1''	1'	1''		1' 1''
4	1' 1''	1'	1'' 1'		1' 1''	1'' 1'	1' 1''
6	1'' 1'	1'		1'' 1'	1'' 1'		1'' 1'

б

	2	4	6	8	10	12	14
2							
4							
6		1'				1'	

ж

	2	4	6	8	10	12	14
2					1'		
4		1'				1'	
6					1'		

в

	2	4	6	8	10	12	14
2			1''	1''			1''
4	1''			1''	1'' 1''		
6	1'' 1''	1''	1''	1''	1'' 1''	1''	

з

	2	4	6	8	10	12	14
2					1'		
4	1''	1'	1''			1''	1''
6		1''			1'	1''	

г

	2	4	6	8	10	12	14
2							1'
4		1'					1''
6			1''				

и

	2	4	6	8	10	12	14
2							1'
4			1'				
6							

д

	2	4	6	8	10	12	14
2							
4							
6							

к

Рис. 1.42

Составляющие рисунка 1.42:

- а – результат опроса А на $\Lambda_{11} \rightarrow X'$;
- б – результат опроса А на $(\Lambda_{11} \vee \Lambda_{21}) \rightarrow (X' \vee X'')$;
- в – результат опроса $(X' \vee X'')$ на $\Gamma = ||1' - 1''||$ (опроса А на $\Lambda_{11}\Lambda_{21}) \rightarrow$ новая матрица X' ;
- г – результат опроса А на $(\Lambda_{11}\Lambda_{21} \vee \Lambda_{31}) \rightarrow$
новая матрица $(X' \vee X'')$;
- д – результат опроса $(X' \vee X'')$ на Γ (опроса А на $\Lambda_{11}\Lambda_{21}\Lambda_{31}) \rightarrow X' \rightarrow Y'$;
- е – результат опроса А на $(\Lambda_{12} \vee \Lambda_{22}) \rightarrow (X' \vee X'')$;
- ж – результат опроса А на $\Lambda_{12}\Lambda_{22} \rightarrow X'$;
- з – результат опроса А на $(\Lambda_{12}\Lambda_{22} \vee \Lambda_{32}) \rightarrow (X' \vee X'')$, $1^* = 1' \vee 1''$;
- и – результат опроса А на $(\Lambda_{11}\Lambda_{21}\Lambda_{31} \vee \Lambda_{12}\Lambda_{22}\Lambda_{32}) \rightarrow (Y' \vee Y'')$;
- к – результат опроса $(Y' \vee Y'')$ на Γ^T (опроса А на $X) \rightarrow Y' \rightarrow$
конечный результат распознавания.

Структура спецпроцессора-идентификатора. Для реализации параллельного алгоритма необходимо образовать обрабатывающий массив (рис. 1.43,а) из буферной памяти (БФ_q – q-й слой буфера) и трех операционных матриц: базовая матрица А – структура однократного распознавания фрагментов Λ_{su} ; В – среда формирования матриц X' и $(X' \vee X'')$ и распознавания строчных фрагментов Γ ; С – среда формирования матриц Y' и $(Y' \vee Y'')$ и распознавания столбцовых фрагментов Γ^T .

Массив функционирует следующим образом. Анализируемый кадр размещается в элементах памяти матрицы А. По сигналам ОПРОС А,В,С результаты опроса соответствующих сред – $X=||x_{ji}||$, $Y=||y_{ji}||$, $Z=||z_{ji}||$ – записываются в БФ_q. При подаче сигнала синхронизации СИ_{В,С} происходит переключение матрицы В или С в соответствии с информацией $V=||v_{ji}||$, записанной ранее в БФ_q. Одновременно формируются результаты очередного опроса на Γ или Γ^T .

Сигналы УСТ В,С подаются перед началом распознавания. Они устанавливают элементы памяти сред В и С в исходное состояние. В шаге 2 алгоритма необходимо предусмотреть дополнительную установку матрицы В. По сигналу ВЫДАЧА из БФ_q считывается результат.

Матрицы $X'(Y')$ и $X''(Y'')$ различаются "окраской" содержащихся в них единиц ($1'$ и $1''$), что достигается использованием в средах В и С специальных элементов памяти.

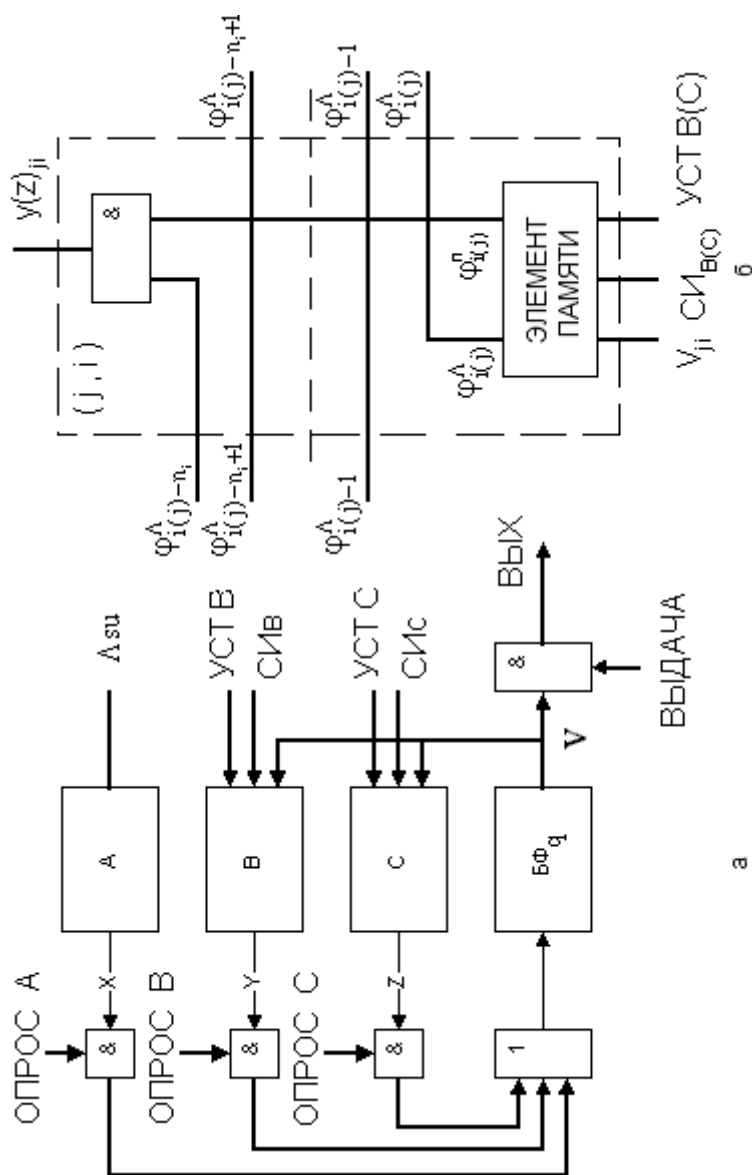


Рис. 1.43. Организация обрабатывающей части чиппроцессора (а) и элемента среды (б)

Значения сигналов на выходах этих элементов: 00 – если данный элемент не является границей ни одного из двух фрагментов, последовательно распознаваемых предыдущей средой; 10, 01 и 11 – если данный элемент является границей только первого из этих фрагментов, только второго и обоих фрагментов соответственно.

Согласно алгоритму, матрица $X'(Y')$ в среде $B(C)$ формируется по каждому нечетному синхроимпульсу, а матрица $X''(Y'')$ – по каждому четному. Это отражено в таблице переходов рассматриваемых элементов (табл. 1.2), где v – соответствующая компонента матрицы V ; s^k – состояние; $\phi_1 \phi_2^k$ – выходы.

При составлении таблицы учтено, что для принятой кодировки выходов в матрицах B или C распознаются фрагменты $\Phi = \parallel (1-)(- -) \dots (- -)(-1) \parallel$ либо Φ^T с двухбитными компонентами. Поэтому, если после четного синхроимпульса на выходах элемента памяти имеем 00 либо 10, то очередное значение $v = 0$.

Организация элемента (j,i) среды $B(C)$ показана на рис. 1.43,б. Входные сигналы на границе среды – нулевые. Через $\Phi_{i(j)}^{\Lambda}, \Phi_{i(j)}^{\Pi}$ обозначены левый и правый выходы элемента памяти.

Таблица 1.2

$s^{k-1} \backslash V$	0	1
1	2, 00	3, 10
2	2, 00	1, 01
3	1, 10	1, 11

$s^k, \phi_1 \phi_2^k$

Функционирование комплекса. Покажем теперь, каким образом проводится на комплексе (базовая ЭВМ – спецпроцессор) идентификация объектов изображения в целом по заданному набору эталонов. Исходное изображение, точнее – его непрерывная развертка по двоичным строкам слева направо и сверху вниз, занимает байты памяти базовой ЭВМ $I = I_0 \dots (I_0 + KL/8 - 1)$. Пусть (y, x) – координаты кадра, $y = 0 \dots (P-1)$; $x = 0 \dots (Q-1)$; величина $j = 0 \dots (k-1)$ – номер строки внутри кадра. Тогда адреса крайних левых байтов кадра

$$I_{yx}^j = I_0 + y \frac{(k-m)L}{8} + x \left[\frac{l-n}{8} \right] + j \left(\frac{L}{8} \right).$$

Массив результатов формирования занимает байты памяти J с базовым адресом J_0 . При каждом (y, x, j) пересылка из одной области памяти в другую идет по $\ell/8$ байт. Поэтому адреса левых крайних байтов кадра (y, x) в области результата формирования

$$J_{yx}^j = J_0 + y \left(\frac{Qkl}{8} \right) + x \left(\frac{kl}{8} \right) + j \left(\frac{l}{8} \right).$$

Приведенные выражения определяют *алгоритм формирования*.

Сформированные кадры $r = 0 \dots (PQ - 1)$ размещаются в буферной памяти спецпроцессора, по одному слою $БФ_r$ на каждый кадр r . Информация о троичных эталонах хранится в оперативной памяти спецпроцессора в байтах $S = S_0 \dots \left(S_0 + \frac{mn}{4} \gamma - 1 \right)$. При этом

каждый троичный фрагмент Λ_{su} занимает ровно 1 байт: 4 разряда – на фрагмент двоичного эталона и еще 4 разряда – на соответствующую маску (случай $n_1=2$). Адресация байтов:

$$S_{su}^t = S_0 + \left(\frac{mn}{4} \right) t + \left(\frac{n}{2} \right) s + u; t = 0 \dots (\gamma - 1); s = 0 \dots \left(\frac{m}{2} - 1 \right); u = 0 \dots \left(\frac{n}{2} - 1 \right).$$

Имеется байтовый регистр, в который последовательно заносятся фрагменты Λ_{su} . В буфере выделяется рабочий слой $БФ_q$. Он постоянно используется для получения промежуточных результатов распознавания в виде единичных отметок в соответствующих битах рассматриваемого в данный момент кадра по каждому типу объекта (рис. 1.42, 1.43,а). Эти результаты передаются в координатный блок, который функционирует параллельно с обрабатывающей частью спецпроцессора (как показал анализ, совмещение имеет место при $m, n \geq 14$).

Блок служит для определения координат идентифицированных объектов. Он содержит простейшую операционную матрицу $k \times \ell$ бит, выполняющую поиск в строках на "≠0", с регистровым обрамлением и микропрограммное управление. Итоговая информация $/t, r, j, i/$ о каждом факте идентификации заносится в оперативную память спецпроцессора.

Приведенные замечания вместе с рассмотренным ранее многотактным алгоритмом определяют структуру и программу работы спецпроцессора в целом.

Пусть N_n – число выполняемых машинных команд при решении рассматриваемой задачи на последовательной ЭВМ, N_ϕ и N_i – то же для комплекса на этапах формирования и идентификации; T_n , T_c , T_ϕ и T_i – соответствующие временные затраты. Сравнительные оценки быстродействия последовательной ЭВМ и комплекса могут быть получены, если известно, за какое число машинных тактов (τ_n , τ_ϕ и τ_i соответственно) выполняется в среднем одна команда в том или иной случае при неизменной длительности такта. Тогда искомая оценка определена отношением T_n/T_c , где $T_n = \tau_n N_n$, $T_c = T_\phi + T_i$, $T_\phi = \tau_\phi N_\phi$, $T_i = \tau_i N_i$. С учетом состава выполняемых команд и структурных особенностей рассматриваемых устройств для проанализированных вариантов было найдено: $\tau_n = 3,45$; $\tau_\phi = 5,25$; $\tau_i = 1,26$.

Результаты проведенных расчетов даны в таблице 1.3.

Таблица 1.3

$K = L$	$m = n$	γ	N_n	N_ϕ	N_i	T_n/T_c
1008	24	64	307539017	256554	8115803	91,68
1008	24	32	164419017	256554	4058203	87,80
504	24	64	77754828	64154	2028953	92,72
1008	16	64	348086506	207814	3069417	242,0

Нетрудно заметить, что эффективность использования спецпроцессора растет с уменьшением размеров объектов. Размеры изображения в целом и параметр γ слабо влияют на эффективность. При этом быстродействие повышается примерно на 2 порядка по сравнению с последовательной ЭВМ с той же длительностью такта.

По условию обработка информации в матрицах A , B и C , как и обращение (чтение - запись) к любому слою буфера, занимают один такт спецпроцессора. В процессе сравнительного анализа быстродействия предполагалось, что в последовательной ЭВМ регистровые команды также выполняются за один такт той же длительности.

Приведенные оценки показывают достаточно высокую эффективность использования спецпроцессора для решения рассмотренной задачи идентификации. Правда, полученные оценки связывались только с производительностью. Для полноты картины необходимо, хотя бы приближенно, оценить дополнительные затраты оборудования. В связи с этим заметим, что при $k = \ell = 128$ и $n_1 = 2$

реализация трех операционных матриц (А, В, С) требует 2294 корпусов БИС на 60 выводов. Использование кристаллов ПЛИС на 264 сигнальных вывода позволит снизить необходимое число корпусов до 406.

Практика 4. Моделирование 4

Для начала моделирования необходимо запустить файл `сruid2.exe`. Моделирование состоит из основной (одиночное распознавание) и дополнительной (множественное распознавание) частей.

Необходимая справка. В основной части предполагается подробное изучение процесса идентификации объекта на кадре. Изучение может проводиться как в пошаговом режиме, так и в нормальном. В пошаговом режиме предлагается посмотреть выполнение процесса распознавания на уровне элементов ЛЗС. Здесь показывается процесс прохождения сигналов во времени через элементы среды. Входы и выходы элементов окрашиваются в синий или красный цвет, что соответствует логическим 0 или 1.

В нормальном режиме программа моделирует работу структуры распознавания, не показывая самого процесса. Программа тестирует моделируемую структуру на правильность функционирования, давая возможность студенту подтвердить на практике изученный ранее теоретический материал.

В дополнительной части демонстрируются возможности алгоритма при множественном распознавании объектов. Объекты загружаются с диска, и по их табличному описанию генерируется кадр, в котором будет проводиться распознавание. При правильном распознавании итоговый результат должен совпасть с заданной таблицей.

Общий вид окна после запуска программы представлен на рис. 1.44. В верхней части окна находятся управляющие пункты меню, а также три панели инструментов, дублирующие пункты меню для облегчения работы. В правой нижней части экрана отображаются имена загруженных кадра, объекта и маски.

Изначально в окне ничего не выводится и большинство пунктов меню недоступно. Их активизация происходит после выбора режима работы - одиночного или множественного распознавания.

Назначение пунктов меню рис. 1.44:

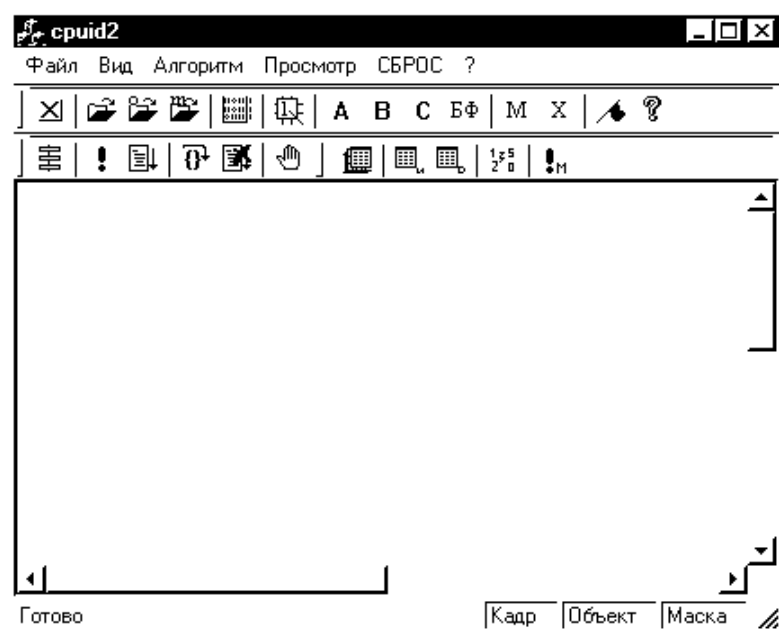


Рис. 1.44

Файл

Загрузка кадра	
Загрузка объекта	Загрузка соответствующих матриц с диска для однозначного распознавания
Загрузка маски	
Загрузить объекты	Загрузка объектов для множественного распознавания
Выход	Выход из программы

Вид

Панель инструментов	Убирает/показывает панель инструментов
Строка статуса	Убирает/показывает строку статуса внизу окна программы.
Панель инструментов однократного распознавания	Убирает/показывает панель инструментов
Панель инструментов множественного распознавания	Убирает/показывает панель инструментов

Алгоритм

Последовательный алгоритм	Запуск алгоритма, реализованного по методу обычного перебора значений ячеек и их сравнения
Параллельный алгоритм	Запуск алгоритма работы спецпроцессора-идентификатора
Пошаговый алгоритм	То же, но в пошаговом режиме
Точки останова	Задание точек приостановки работы в процессе выполнения алгоритма. Необходимо, если нужно просмотреть состояние матриц на каком-либо шаге алгоритма
Следующий шаг	Запуск следующего шага в случае, если произошел останов на заданном шаге
Остановить алгоритм	Прекращение работы алгоритма в пошаговом режиме
Генерировать кадр	Генерация кадра по табличному описанию объектов
Таблица с исходными координатами	Вывод на экран таблицы с заданными координатами локализаций объектов
Таблица с распознанными координатами	Вывод на экран таблицы с распознанными координатами локализаций объектов
Запуск алгоритма множественного распознавания	Запуск алгоритма множественного распознавания для сгенерированного кадра

Просмотр

Показать результат	Показать результат распознавания объекта на кадре
Элементы матрицы	Позволяет посмотреть внутренние состояния элементов матриц и значения их выводов в условном графическом изображении (рис. 1.46).

Матрица А	
Матрица В	
Матрица С	Вывод на экран содержимого соответствующих матриц
Буфер БФq	
Маска	
Объект	
Сброс	Сброс всех результатов и обнуление загруженных матриц

Способ задания матриц. Для режима однократного распознавания используется следующий формат матриц. Матрицы представляют собой текстовый файл формата ASCII. Программа производит последовательное чтение символов из файла и преобразует их в числовой вид. Знак перевода строки означает конец строки. Количество столбцов считается по количеству символов в строке. Для редактирования значений ячеек достаточно открыть файл в любом текстовом редакторе DOS. Данные, записанные в этом файле, выглядят так, как будто загружены в память. Поэтому следует быть осторожным при ручном редактировании файла.

Для режима многотактного распознавания, кроме текстовых объектов формата ASCII, возможна загрузка объектов черно-белых битовых матриц с расширением (*.bmp). Данные файлы могут быть созданы в любом графическом редакторе типа PaintBrush.

Размеры матриц должны удовлетворять условиям табл. 1.4.

Таблица 1.4

Матрица	Одиночное распознавание	Множественное распознавание
Кадр	128×128 (максимально)	1000×1000 (фиксировано)
Объект	(2-128)×(2-128)	20×20 (фиксировано)
Маска	Должна совпадать с размером объекта	

Числа строк и столбцов матриц должны быть четными.

Основная часть (одиночное распознавание). Для начала работы в режиме одиночного распознавания необходимо загрузить кадр, объект и маску объекта. Это делается выбором соответствующих пунктов меню "Файл" или нажатием аналогичных кнопок на панели инструментов.

После загрузки указанных данных становится возможным просмотр загруженных матриц и запуск алгоритма распознавания.

Просмотр матриц осуществляется следующим образом.

При нажатии на кнопки А, В, С, БФ, М и Х на экране отображаются соответственно содержимое трех матриц А,В,С обрабатывающей части спецпроцессора, буферной памяти, маски объекта и содержимое самого загруженного объекта.

При нажатой кнопке "Просмотр элементов" возможен просмотр матриц А,В,С как набора логических элементов (рис. 1.45).

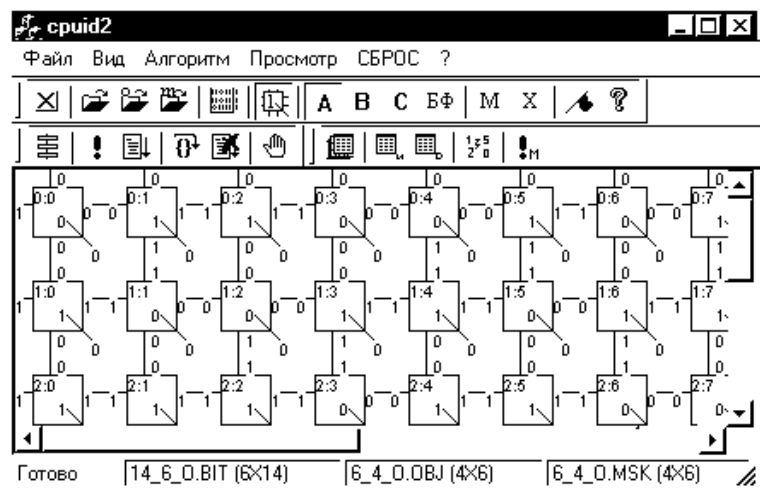


Рис. 1.45

Каждый элемент индицирует свое состояние соответствующей окраской рамки и цифрой 0 или 1 внутри. Элементы связаны между собой проводниками, которые окрашиваются в зависимости от уровня проходящих по ним сигналов: "Лог. 0" – синий цвет, "лог. 1" – красный цвет. На элементе также отображен его номер (j,i) в матрице.

Запуск алгоритма возможен как в пошаговом режиме, так и в нормальном. В нормальном режиме после окончания распознавания станет доступным пункт меню "Просмотр результата", при выборе которого на экране отобразится матрица А с распознанными на ней объектами (рис. 1.46).

При работе в пошаговом режиме есть возможность просмотреть процесс функционирования обрабатывающей части спецпроцессора на любом шаге алгоритма. Для этого необходимо расста-

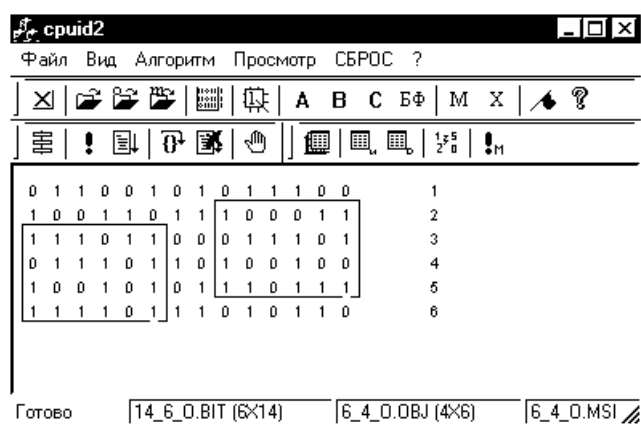


Рис. 1.46

вить контрольные точки. Диалог расстановки контрольных точек запускается при выборе соответствующего пункта меню (рис. 1.47).

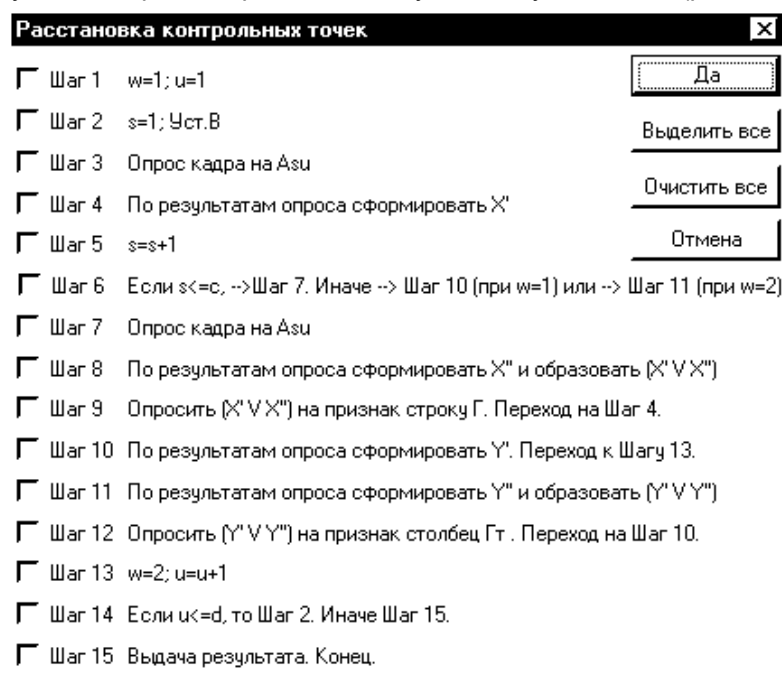


Рис. 1.47

После расстановки контрольных точек в нужных пунктах алгоритма необходимо осуществить запуск пошагового алгоритма нажатием кнопки или через пункт меню. Программа автоматически дойдет до контрольного пункта и остановится, ожидая запуска следующего шага.

Для перехода к следующей контрольной точке необходимо выбрать пункт меню "Следующий шаг". Пока не выбран следующий шаг, у пользователя есть возможность просмотреть состояния матриц на данный момент.

По завершении работы алгоритма выдается сообщение о его окончании, и пользователь может посмотреть конечный результат распознавания.

Если в процессе выполнения алгоритма в пошаговом режиме возникла необходимость закончить алгоритм, то в меню нужно выбрать пункт "Остановить алгоритм".

Дополнительная часть (множественное распознавание). Для перехода в режим множественного распознавания сначала сбрасываются все установки внутренних переменных путем выбора пункта меню "Сброс". Затем выполняется загрузка объектов, для чего необходимо выбрать пункт меню "Загрузка объектов". На экране появится диалог (рис. 1.48).

Диалог условно разделен на две половины. В левой половине располагается список известных программе файлов с объектами. В правой половине находится список объектов, которые по заданным координатам будут генерироваться в кадр.

Список известных файлов можно редактировать с помощью трех кнопок:

Новый	Добавление новых файлов к списку
Удалить	Удаление выбранного файла из списка
Запомнить список	Запоминание текущего списка

Чтобы добавить выбранный объект в список используемых файлов, необходимо нажать кнопку "Добавить>>". Выбранный файл появится в списке используемых. Этому файлу можно присвоить любое имя, под которым он условно будет обозначен в таблице координат, и назначить ему количество локализаций в кадре.

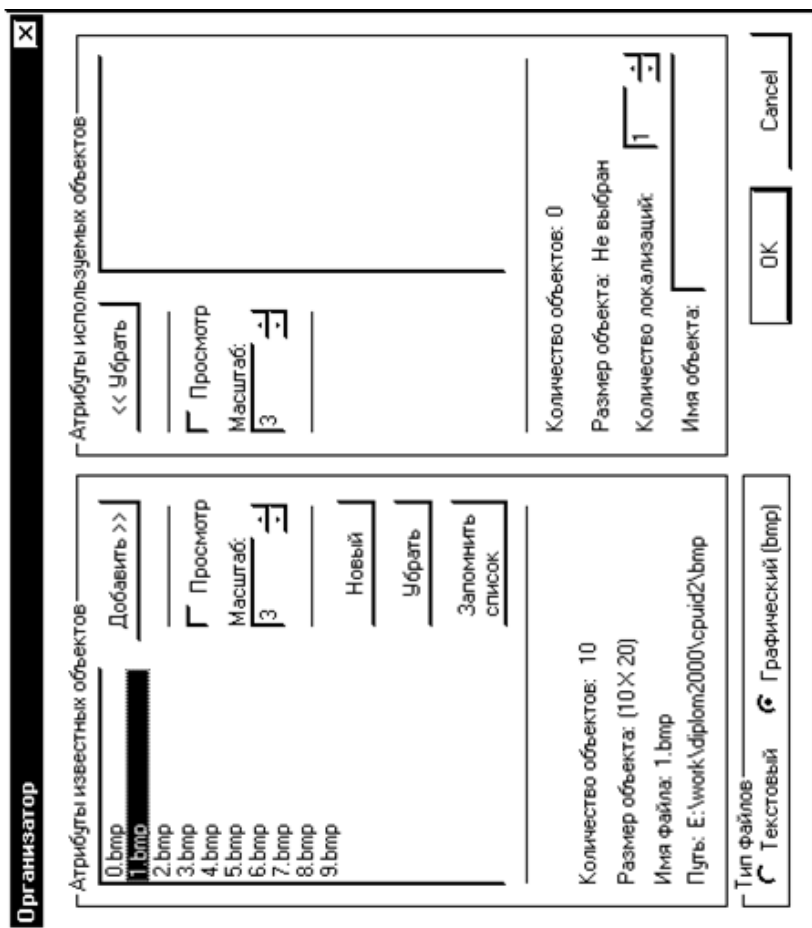


Рис. 1.48

При выборе необходимого количества объектов и нажатии кнопки ОК на экране появится диалог с таблицей координат локализаций объектов (рис. 1.49).

В этом диалоге представлены имена загруженных объектов со случайно сгенерированными координатами. Для того, чтобы исключить совпадение координат и нулевые координаты, необходимо нажать кнопку "Контроль координат". Кнопкой "Расставить по узлам" координаты вновь случайно генерируются в узлах сетки (40×40) на кадре 1000×1000.

Ввод исходных данных для таблицы объектов						
Таблица объектов		Координаты локализаций объектов				
№	Имя	1	2	3	4	5
1	7	(400 x 760)	(160 x 200)	(640 x 80)	нет	нет
2	3	(440 x 320)	(960 x 440)	(680 x 320)	нет	нет

◀
▶

Рис. 1.49

При необходимости можно вручную изменить значение координат. Для этого необходимо сделать двойной щелчок мыши по номеру объекта. После нажатия кнопки ОК необходимо сгенерировать кадр, выбрав пункт меню "Генерировать кадр".

После генерации кадра нужно выбрать пункт меню "Запуск алгоритма множественного распознавания". По окончании его работы на экране появится сообщение и диалог с таблицей распознанных координат (рис. 1.50).

Вывод данных						
Таблица объектов		Координаты локализаций объектов				
№	Имя	1	2	3	4	5
1	3	(600 x 40)	(100 x 100)	(40 x 200)	(200 x 280)	нет
2	2	(160 x 40)	(520 x 880)	нет	нет	нет

◀
▶

Рис. 1.50

Примечание. Процесс моделирования для каждой части занятия следует повторить несколько раз при различных исходных данных. Правильность получаемых результатов будет свидетельствовать об адекватности рассмотренного ранее параллельного алгоритма распознавания принятой структуре спец-процессора.

ЛИТЕРАТУРА К РАЗДЕЛУ I

1. Райхлин В.А. Основы цифровой схемотехники. – Казань: Изд. КГТУ. 2006.
2. Райхлин В.А. Операционные логико-запоминающие среды. Вопросы применения и синтеза //Автоматика и телемеханика.1983. № 11.
3. Райхлин В.А. Об использовании аппарата двумерного ассоциативного поиска в процессе распознавания /Проблемно-ориентированные средства повышения эффективности вычислительных систем. – Казань: Изд. КГТУ, 1991.
4. Варшавский В.И. и др. Однородные структуры. – М.: Энергия. 1973.
5. Каутц У.Х. Однородные логико-запоминающие среды и большие интегральные схемы /Синтез автоматов и управление на сетях связи. - М.: Наука. 1973.
6. Ульман Дж.Д. Вычислительные аспекты СБИС. – М.: Радио и связь, 1990

РАЗДЕЛ II. ПАРАЛЛЕЛЬНЫЕ СУБД

Введение

РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ. ОСОБЕННОСТИ ПАРАЛЛЕЛЬНЫХ СУБД

Процессы обработки структурированных и неструктурированных данных (медиа данные) в больших объемах немыслимы без использования современных параллельных систем управления базами данных (СУБД). На сегодняшний день существует около десятка коммерческих систем данного класса: Oracle RAC, NCR Teradata, IBM Informix OnLine, IBM DB2, Microsoft SQL Server и др. Как правило, подобные системы функционируют на параллельных архитектурах класса мейнфрейм (mainframe). Объемы внешней памяти таких систем измеряются десятками, сотнями и даже тысячами гигабайт. Стоимость подобных систем также впечатляет.

Вместе с тем появляются и некоммерческие программные продукты параллельной обработки данных. Они разрабатываются как открытые системы, предоставляющие исходные коды программного обеспечения под лицензией GNU. К числу таких систем можно отнести СУБД MySQL Cluster и PostgreSQL Cluster. Они представляют собой полноценные СУБД кластерного типа, реализующие все стандартные функции. Требования к аппаратному обеспечению таких систем значительно скромнее. Скромнее и возможности, но их вполне хватает для построения высокопроизводительных систем обработки информации среднего предприятия. Подобные системы могут быть созданы например на базе имеющихся компьютерных классов с привлечением небольшой группы специалистов.

Реляционная модель данных [1, 2]

Реляционная модель предложена сотрудником компании IBM Е.Ф.Коддом в 1970 г. В настоящее время эта модель является фактическим стандартом, на который ориентируются практически все современные коммерческие СУБД. Дадим определения некоторым базовым понятиям.

Декартово произведение: Для заданных конечных множеств D_1, D_2, \dots, D_n (не обязательно различных) декартовым произведением $D_1 * D_2 * \dots * D_n$ называется множество произведений вида $d_1 * d_2 * \dots * d_n$, где $d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n$.

Пример. Пусть заданы два множества – $A(a_1, a_2, a_3)$ и $B(b_1, b_2)$. Тогда их декартово произведение

$$C = A * B(a_1 * b_1, a_2 * b_1, a_3 * b_1, a_1 * b_2, a_2 * b_2, a_3 * b_2).$$

Отношение. Отношением R , определенным на множествах D_1, D_2, \dots, D_n называется подмножество декартова произведения $D_1 * D_2 * \dots * D_n$. При этом:

- Множества D_1, D_2, \dots, D_n называются *доменами отношения*.
- Элементы декартова произведения $d_1 * d_2 * \dots * d_n$ называются *кортежами*.
- Число n определяет степень, или «арность» отношения ($n=1$ – унарное, $n=2$ – бинарное, ..., в общем случае – n -арное).
- Количество кортежей называется *мощностью отношения*.

Пример. На множестве C из предыдущего примера могут быть определены отношения $R_1(a_1 * b_1, a_3 * b_2)$ или $R_2(a_1 * b_1, a_2 * b_1, a_1 * b_2)$.

Отношения удобно представлять в виде таблиц. На рис. 2.1 представлена таблица (отношение степени 5), содержащая некоторые сведения о работниках гипотетического предприятия.

Отношение →	целое		строка		целое	← Типы данных
	номер		имя	должность	деньги	← Домены
	табельный номер	имя	должность	оклад	премия	← Атрибуты
	2930	Иванов	инженер	112	40	← Кортежи
	2931	Петров	вед. инженер	144	50	
	2932	Сидоров	бухгалтер	92	35	
	↑ Ключ					

Рис. 2.1

Строки таблицы соответствуют кортежам. Каждая строка представляет собой описание одного объекта реального мира (в данном случае – работника), характеристики которого содержатся в столбцах. Можно провести аналогию между элементами реляционной модели данных и элементами модели «сущность-связь». Реляционные отношения соответствуют наборам сущностей, а кортежи – сущностям. Поэтому, как и в модели «сущность-связь», столбцы в

таблице, представляющей реляционное отношение, называют *атрибутами*.

Каждый атрибут определен на домене. Поэтому домен можно рассматривать как множество допустимых значений данного атрибута.

Несколько атрибутов одного отношения и даже атрибуты разных отношений могут быть определены на одном и том же домене. В примере, показанном на рис. 2.1, атрибуты "Оклад" и "Премия" определены на домене "Деньги". Поэтому понятие домена имеет семантическую нагрузку: данные можно считать сравнимыми только тогда, когда они относятся к одному домену. Так, в рассматриваемом нами примере сравнение атрибутов "Табельный номер" и "Оклад" является семантически некорректным, хотя они и содержат данные одного типа.

Именованное множество пар "имя атрибута – имя домена" называется *схемой отношения*. Набор именованных схем отношений представляет собой *схему базы данных*.

Атрибут, значения которого однозначно идентифицирует кортежи, называется *ключевым* (или просто *ключом*). В нашем случае ключом является атрибут "Табельный номер", поскольку его значение уникально для каждого работника предприятия. Если кортежи идентифицируются только сцеплением значений нескольких атрибутов, то говорят, что отношение имеет составной ключ.

Отношение может содержать несколько ключей. Но всегда один из ключей объявляется *первичным*, его значения не могут обновляться. Все остальные ключи отношения называются *возможными ключами*.

В отличие от иерархической и сетевой моделей данных, в реляционной модели отсутствует понятие группового отношения. Для отражения ассоциаций между кортежами разных отношений используется дублирование их ключей (рис. 2.2). Например связь между отношениями ОТДЕЛ и СОТРУДНИК создается путем копирования первичного ключа "Номер_отдела" из первого отношения во второе.

Таким образом:

- Для того, чтобы получить список работников данного подразделения, необходимо:

1. Из таблицы ОТДЕЛ установить значение атрибута "Номер_отдела", соответствующее данному "Наименованию_отдела".

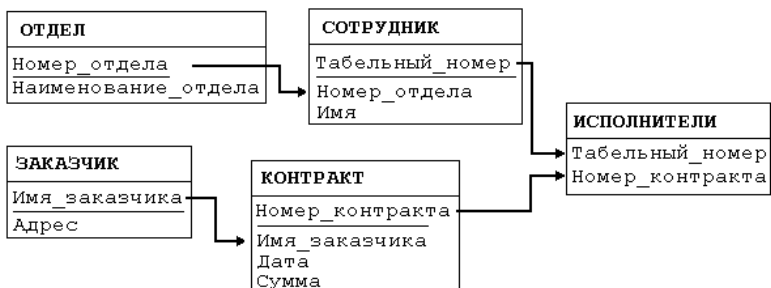


Рис.2.2

2. Выбрать из таблицы СОТРУДНИК все записи, значение атрибута "Номер_отдела" которых равно полученному на предыдущем шаге.

- Для того, чтобы узнать в каком отделе работает сотрудник, нужно выполнить обратную операцию:

1. Определяем "Номер_отдела" из таблицы СОТРУДНИК.
2. По полученному значению находим запись в таблице ОТДЕЛ.

Атрибуты, представляющие собой копии ключей других отношений, называются *внешними ключами*.

Свойства отношений:

1. Отсутствие кортежей-дубликатов. Из этого свойства вытекает наличие у каждого кортежа первичного ключа. Для каждого отношения полный набор его атрибутов является первичным ключом. Однако при определении первичного ключа должно соблюдаться требование "минимальности", т.е. в такой ключ не должны входить те атрибуты, которые можно отбросить без ущерба для основного свойства первичного ключа – однозначно определять кортеж.

2. Отсутствие упорядоченности кортежей.

3. Отсутствие упорядоченности атрибутов. Для ссылки на значение атрибута всегда используется имя атрибута.

4. Атомарность значений атрибутов: среди значений домена не могут содержаться множества значений, т.е. отношения.

Операции над данными (реляционная алгебра)

Различают обработку кортежей и обработку отношений.

Операции обработки кортежей связаны с изменением состава кортежей в каком-либо отношении. К ним относятся:

ДОБАВИТЬ – надо задать имя отношения и ключ кортежа.

УДАЛИТЬ – необходимо указать имя отношения и идентифицировать кортеж или группу кортежей, подлежащих удалению.

ИЗМЕНИТЬ – выполняется для названного отношения и может корректировать как один, так и несколько кортежей.

Операции обработки отношений. На входе каждой такой операции используется одно или несколько отношений. Результатом выполнения операции всегда является новое отношение.

В рассматриваемых ниже примерах (они заимствованы из [1]) используются следующие отношения:

P (D1, D2, D3)	Q (D4, D5)	R (M, P, Q, T)	S (A, B)
1 11 x	x 1	x 101 5 a	5 a
2 11 y	x 2	y 105 3 a	10 b
3 11 z	y 1	z 500 9 a	15 c
4 12 x		w 50 1 b	2 d
		w 10 2 b	6 a
		w 300 4 b	1 b

В реляционной алгебре определены следующие операций обработки отношений:

Проекция /PROJECT/. Суть этой операции состоит в том, что берется отношение R, удаляются некоторые из его компонентов и (или) переупорядочиваются оставшиеся компоненты.

Обозначение: $\pi_A(R)$. Пример:

$$\pi_{M,T} = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \\ w & b \\ w & b \end{bmatrix} = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix}$$

Выборка /SELECT/. На входе используется одно отношение, результат - новое отношение, построенное по той же схеме, содержащее подмножество кортежей исходного отношения, удовлетворяющих условию выборки.

Обозначение: $\sigma_{\theta}(R)$, θ - условие селекции. Пример:

$$\sigma_{\hat{A}=11}(R) = \begin{bmatrix} 1 & 11 & x \\ 2 & 11 & y \\ 3 & 11 & z \end{bmatrix}$$

Объединение /UNION/. Отношения-операнды в этом случае должны быть определены по одной схеме. Результирующее отношение содержит все строки операндов, за исключением повторяющихся. Обозначение: $R_1 \cup R_2$. Пример:

$$\sigma_{Q,T}(R) \cup S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cup \begin{bmatrix} 5 & a \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \\ 10 & b \\ 15 & c \\ 6 & a \end{bmatrix}$$

Пересечение /INTERSECT/. На входе операции – два отношения, определенные по одной схеме. На выходе – отношение, содержащие кортежи, которые присутствуют в обоих исходных отношениях. Обозначение: $R_1 \cap R_2$. Пример:

$$\sigma_{Q,T}(R) \cap S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cap \begin{bmatrix} 5 & a \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix}$$

Разность. Операция, во многом похожая на Пересечение, за исключением того, что в результирующем отношении содержатся кортежи, присутствующие в первом и отсутствующие во втором исходных отношениях. Обозначение: $R_1 - R_2$. Пример:

$$\sigma_{Q,T}(R) - S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} - \begin{bmatrix} 5 & a \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} 3 & a \\ 9 & a \\ 2 & b \\ 4 & b \end{bmatrix}$$

Декартово произведение. Входные отношения могут быть определены по разным схемам. Схема результирующего отношения включает все атрибуты исходных. Кроме того:

- степень результирующего отношения равна сумме степеней исходных отношений
- мощность результирующего отношения равна произведению мощностей исходных отношений. Обозначение: $R_1 \times R_2$. Пример:

$$\sigma_{M,T}(R) \times (\sigma_{Q,T}(R) \cap S) = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix} \times \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} x & a & 5 & a \\ x & a & 1 & b \\ y & a & 5 & a \\ y & a & 1 & b \\ z & a & 5 & a \\ z & a & 1 & b \\ w & b & 5 & a \\ w & b & 1 & b \end{bmatrix}$$

Соединение /JOIN/. Эта операция имеет сходство с Декартовым произведением. Но здесь добавлено условие, согласно которому, вместо полного произведения всех строк, в результирующее отношение включаются только строки, удовлетворяющие определенному условию θ между атрибутами соединяемых отношений. Обозначение: $\pi_A(\sigma_\theta(R \times S))$. Пример:

$$\pi_{D1,D2,D3,D5}(\sigma_{D3=D4}(P \times Q)) = \begin{bmatrix} 1 & 11 & x & 1 \\ 1 & 11 & x & 2 \\ 2 & 11 & y & 1 \\ 4 & 12 & x & 1 \\ 4 & 12 & x & 2 \end{bmatrix}$$

Свойства параллельных систем [4]

Идеальная параллельная система обладает двумя главными свойствами: *линейное ускорение* (вдвое большее аппаратное обеспечение выполнит ту же задачу в два раза быстрее) и *линейная расширяемость* (вдвое большее аппаратное обеспечение выполнит вдвое большую задачу за то же время).

Более формально, если одна и та же работа выполняется на меньшей и на в N раз большей системе, то увеличение быстродействия (*ускорение* R), даваемое большей системой, определится как

$$R = T_1 / T_2.$$

Здесь T_1 – время, затраченное меньшей системой, T_2 – время, за-

траченное большей системой. Для линейного ускорения $R = N$. Ускорение позволяет определить эффективность наращивания системы на сопоставимых задачах.

Коэффициент расширяемости M определяется как

$$M = T_1' / T_2',$$

где T_1' – время, затраченное меньшей системой на решение небольшой задачи; T_2' – время, затраченное в N раз большей системой на решение в N раз большей задачи. Если $M = 1$, то расширяемость называется линейной. Расширяемость позволяет оценить эффективность наращивания системы на больших задачах.

Существуют два различных вида расширяемости: *пакетная* и *транзакционная*.

Если суть работы состоит в выполнении большого количества небольших независимых запросов от многих пользователей к базе данных коллективного пользования, то свойство расширяемости состоит в удовлетворении в N раз большего числа запросов от большего в N раз числа клиентов к большей в N раз базе данных. Этот вид расширяемости называется *транзакционным*. Он идеально подходит для параллельных систем, так как каждая транзакция представляет собой небольшую независимую работу, которая может выполняться на отдельном процессоре.

Пакетная расширяемость имеет место, когда задача состоит в выполнении одной большой работы. Она характерна для запросов к базам данных и для задач математического моделирования. В этих случаях расширяемость состоит в использовании в N раз большего компьютера для решения в N раз большей задачи. Для систем баз данных пакетная расширяемость выражается во времени выполнения того же запроса к в N раз более крупной базе данных. Для научных задач – во времени выполнения того же расчета на в N раз более мелкой сетке или для в N раз более трудоемкого моделирования.

Достижимость линейного ускорения и линейного расширения затруднена тремя факторами:

- *Запуск* – время, необходимое для запуска параллельной операции. Если нужно запустить тысячи процессоров, то реальное время вычислений может оказаться значительно меньше времени, требуемого для их запуска.

- *Помехи* – появление каждого нового процесса ведет к замедлению всех остальных процессов,использующих те же ресурсы.
- *Перекоc* – с увеличением числа параллельных шагов средняя продолжительность выполнения каждого шага уменьшается, но отклонение от среднего значения может значительно превзойти само среднее значение. Время выполнения работы – это время выполнения наиболее медленного шага работы. Когда отклонение от средней продолжительности превосходит ее саму, то параллелизм позволяет только слегка убыстрить выполнение работы.

Аппаратная архитектура систем баз данных [4 – 7]

Каким образом должна быть построена расширяемая много-процессорная система баз данных? Стоунбрейкер [4] предложил следующую классификацию для целого спектра разработок (рис. 2.1 – 2.3).

Совместно используемая память (рис.2.1). Все процессоры имеют прямой доступ к общей глобальной памяти и ко всем дискам. Примерами подобных систем являются мультипроцессоры IBM/370, Digital VAX, Sequent Symmetry.



Рис. 2.1

Совместно используемые диски (рис.2.2). Каждый процессор имеет не только свою собственную память, но и прямой доступ ко всем дискам. Примерами являются IBM Sysplex и первоначальная версия Digital VAXcluster.

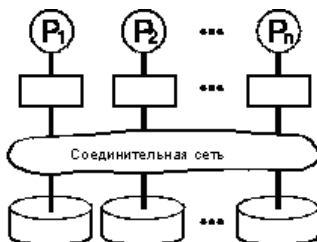


Рис.2.2

Отсутствие совместного использования ресурсов (рис. 2.3). Каждая память и диск находятся в распоряжении какого-либо процессора, который работает как сервер хранящихся в них данных. Массовое запоминающее устройство в таких архитектурах распределено между процессорами посредством соединения одного или более дисков. Примерами таких машин являются Teradata, Tandem и nCUBE.



Рис. 2.3

Архитектуры без совместного использования ресурсов сводят к минимуму помехи посредством минимизации совместно используемых ресурсов. Кроме того, при использовании массово производимых процессоров и памяти им не требуется сверхмощная соединительная сеть. В таких архитектурах через сеть передаются только вопросы и ответы, а не большие массивы данных. Непосредственные обращения к памяти и к дискам обрабатываются локальным процессором, и только отфильтрованные (урезанные) данные передаются запрашивающей программе. Это позволяет реализовать более расширяемую архитектуру за счет минимизации трафика в соединительной сети.

Отсутствие совместного использования ресурсов характерно для систем баз данных, используемых в проектах Teradata, Gamma, Tandem, Bubba, Arbre и nCUBE. Системы для рабочих групп от 3com, Borland, Digital, HP, Novell, Microsoft и Sun также базируются на архитектуре типа клиент-сервер без совместного использования ресурсов.

Реальные соединительные сети, используемые в этих системах, зачастую совершенно не схожи друг с другом. Teradata использует избыточную древовидную соединительную сеть. Tandem – трехуровневую дуплексную сеть: два уровня внутри кластера и

кольца, соединяющие кластеры. Единственное требование, которое Arbre, Bubba и Gamma предъявляют к соединительной сети, состоит в существовании связи между любыми двумя узлами. Gamma работает на Intel Hypercube. Прототип Arbre был реализован на основе процессоров IBM 4381, соединенных друг с другом в сеть напрямую. Системы для рабочих групп переходят с Ethernet на более высокоскоростные локальные сети.

Основным преимуществом мультипроцессоров без совместного использования ресурсов является то, что число процессоров в них может достигать до сотен и даже тысяч без возникновения каких-либо помех в работе одного со стороны другого. Компании Teradata, Tandem и Intel запустили проекты систем с более чем 200 процессорами. Intel разрабатывает гиперкуб с 2000 узлами. Максимальное число процессоров в мультипроцессорной системе с разделением памяти равно к настоящему моменту 32.

Архитектуры без совместного использования ресурсов позволяют достичь почти линейного ускорения и расширяемости на сложных реляционных запросах и при транзакционной обработке запросов.

Вот почему проектировщики машин баз данных не видят смысла в выполнении сложных аппаратных и программных проектов с совместным использованием памяти и дисков.

Системы с совместным использованием памяти и дисков не так-то легко наращивать для приложений, связанных с базами данных. Основная проблема для мультипроцессоров с совместным использованием памяти – помехи. Соединительная сеть должна иметь пропускную способность, равную сумме пропускных способностей процессоров и дисков. Создать сеть, наращиваемую до тысячи узлов, – весьма непростая задача.

Для того, чтобы уменьшить трафик в сети и свести к минимуму время ожидания, каждому процессору придается большая собственная кэш-память. Эксперименты на мультипроцессорах с совместным использованием памяти, выполняющих задачи с базами данных, показывают, что загрузка и выталкивание из кэш-памяти значительно снижают производительность системы. При возраста-

нии параллелизма помехи при совместном использовании ресурсов ограничивают рост производительности.

Для уменьшения помех в многопроцессорных системах часто используется *механизм «родственного» планирования*, предполагающий закрепление каждого процесса за конкретным процессором, что является формой *разделения данных*. Другие процессоры, желающие получить доступ к данным, посылают сообщения к серверам, управляющим этими данными. В системах с совместно используемой памятью разделение данных создает множество проблем, связанных с перекосом и распределением нагрузки. Поэтому при работе с базами данных такие мультипроцессоры могут быть экономично расширены только до нескольких процессоров.

Для борьбы с помехами в них часто применяется *архитектура с совместным использованием дисков*, что является логическим следствием родственного планирования. Если дисковая соединительная сеть наращивается до тысяч дисков и процессоров, то схема с совместным использованием дисков пригодна только для больших баз данных, предназначенных лишь для чтения, и для баз данных без одновременного использования. Эта схема мало эффективна для прикладных программ баз данных, которые считывают и записывают совместно используемые данные.

Если процессору нужно изменить какие-либо данные, он сначала должен получить текущую их копию. Так как другие процессоры могут в это время изменять те же самые данные, то процессор должен заявить о своих намерениях. Он может прочитать совместно используемые данные с диска и изменить их только в случае, если его намерение одобрено всеми остальными процессорами. Тогда они смогут учесть проведенные изменения в своей дальнейшей работе. Имеется множество вариантов оптимизации этого протокола. Но все они сводятся к обмену сообщениями о резервировании данных и обмену большими физическими массивами данных. Это приводит к помехам, задержкам и большому трафику в совместно используемой соединительной сети.

Для прикладных программ с совместно используемыми данными подход с совместным использованием дисков обходится зна-

чительно дороже, чем подход без совместного использования ресурсов с обменом логическими вопросами и ответами высокого уровня между клиентами и серверами.

Такое решение возникло на основе применения *мониторов обработки транзакций*, которые разделяют нагрузку между отдельными серверами. Кроме того, оно основано на механизме вызова удаленных процедур.

Подчеркнем еще раз, что тенденция к разделению данных и к архитектуре без совместного использования ресурсов позволяет уменьшить помехи в системах с совместно используемыми дисками. Поскольку соединительную сеть системы с совместным использованием дисков практически невозможно расширить до тысяч процессоров и дисков, многие сходятся на том, что лучше с самого начала ориентироваться на архитектуру без совместного использования ресурсов.

Выработано единое мнение об архитектуре распределенных и параллельных систем баз данных. Эта архитектура базируется на идее аппаратного обеспечения без совместного использования ресурсов, когда процессоры поддерживают связь друг с другом только посредством передачи сообщений через соединяющую их сеть. В таких системах corteжи каждого отношения в базе данных разделяются между дисковыми запоминающими устройствами, напрямую подсоединенными к каждому процессору. Разделение позволяет нескольким процессорам просматривать большие отношения параллельно, не прибегая к использованию каких-либо экзотических устройств ввода/вывода.

Такая архитектура впервые была представлена компанией Teradata в конце 70-х годов и применена в нескольких исследовательских проектах. Теперь она используется в продуктах Teradata, Tandem, Oracle-nCUBE и еще нескольких продуктах, находящихся в стадии разработки. Исследовательское сообщество использовало архитектуру без совместного использования ресурсов в таких системах, как Arbre, Bubba и Gamma.

Занятие 5.

СУБД MySQL Cluster.

Целью этого занятия является знакомство с архитектурой, принципами работы, обеспечением надежности и основными управляющими командами СУБД MySQL Cluster.

Лекция 5. Начальное знакомство [8]

Архитектура

Общее описание. MySQL Cluster – это параллельная СУБД, которая позволяет кластеризовать базу данных, хранящуюся в памяти системы без разделения ресурсов. Такая система позволяет работать с недорогим оборудованием при минимальных требованиях к аппаратуре и программному обеспечению. MySQL Cluster включает в себя стандартный сервер MySQL и механизм хранения данных, названный NDB, или NDB Cluster.

“Платформой” системы является набор компьютеров. На каждом из них запущены один или более процессов, которые могут включать в себя сервер MySQL, узел данных, управляющий сервер и, возможно, специализированные программы доступа к данным. Взаимосвязи между этими компонентами показаны на рис. 2.4.

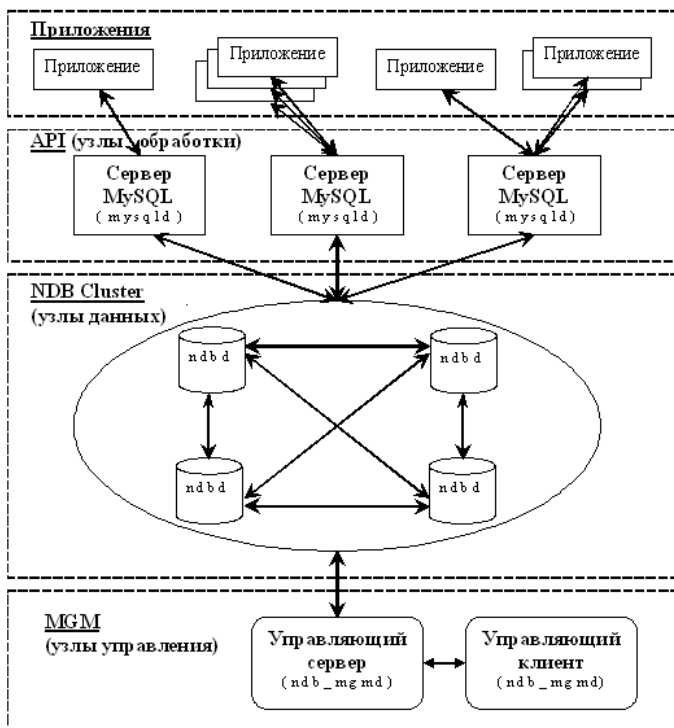


Рис. 2.4

MySQL Cluster состоит из трех типов узлов:

- *Управляющий узел (MGM).* Функции этого типа узла – управление другими узлами в MySQL Cluster, предоставление конфигурационных данных, запуск и остановка узлов, запуск резервирования и т.д. Так как узел типа *MGM* управляет конфигурацией, то управляющие узлы должны быть запущены в первую очередь. *MGM* запускается при помощи команды *ndb_mgmd*.
- *Узел данных (NDB).* Этот тип узлов хранит данные. Необходимое количество таких узлов зависит от числа копий базы данных и от количества фрагментов, на которые необходимо разбить базу. Узлы *NDB* обрабатывают все транзакции.
- *Узел обработки (API).* Узлы типа *API* имеют непосредственный доступ к данным, хранящимся в NDB Cluster. Это обычные сервера

MySQL, настроенные на использование NDB Cluster. MySQL сервер предоставляет стандартный SQL интерфейс. Сервер обрабатывает запросы, преобразуя их транзакции к узлам данных.

Принципы работы

Конфигурирование кластера включает конфигурирование каждого индивидуального узла в кластере и установление индивидуальных коммуникационных связей между узлами. MySQL Cluster разработан, исходя из предположения, что все узла хранения примерно одинаковы по вычислительной мощности, размерам памяти и полосы пропускания.

MySQL Cluster держит все данные в оперативной памяти для транзакций и быстрого восстановления, ограничивая тем самым операции ввода/вывода. Запись на диск производится асинхронно для протоколирования транзакции.

Данные распределены следующим образом. Все таблицы поделены горизонтально на разделы. Каждому разделу отвечает свой узел хранения. Все узлы могут быть поделены на группы. Между узлами одной группы данные зеркалируются. Это делается для повышения надежности системы (вопрос будет рассмотрен далее). Число групп зависит от числа копий базы данных и от общего количества узлов хранения. Например, если имеются 6 узлов хранения и 2 копии базы данных, то число групп равно 3 по два узла хранения в каждой.

Параллельная обработка в MySQL Cluster использует два вида параллелизма:

- межзапросный параллелизм;
- внутрizaпросный параллелизм.

Межзапросный параллелизм означает выполнение множества запросов на независимых API узлах. Его реализация затруднена тем, что MySQL Cluster не содержит утилит для распределения нагрузки между независимыми API.

Внутрizaпросный параллелизм подразумевает разделение данных между узлами хранения данных. Как отмечено ранее, каждый узел хранения содержит в оперативной памяти свой раздел базы данных. Для API узлов, выполняющих основную обработку, узлы данных представляют собой кэш базы данных.

Параллельная архитектура MySQL Cluster позволяет достичь, по словам разработчиков, почти линейной расширяемости. Предположительно, линейная расширяемость может быть достигнута при одновременном увеличении числа узлов хранения и числа API узлов. Но здесь все не так просто, ибо с ростом числа узлов возрастает влияние «накладных расходов» (коллизии в сети, латентность, неравномерное распределение нагрузок по процессорам и др.). Это может существенно влиять на производительность системы. Так что вопрос о масштабируемости MySQL Cluster пока остается открытым.

MySQL Cluster поддерживает следующие сетевые протоколы: TCP/IP и SCI. TCP/IP используется по умолчанию для установления соединения между узлами. Соединения устанавливаются автоматически, исходя из конфигурации кластера. При этом предусмотрена возможность самостоятельной настройки сетевых взаимодействий в кластере.

Обеспечение надежности в MySQL Cluster

Архитектура MySQL Cluster была спроектирована для достижения высокой надежности сервера MySQL. С этой целью:

- API узлы подключены ко всем узлам хранения. Если в одном из узлов хранения произойдет сбой, API узел может использовать другой узел для выполнения транзакции;
- хранимая информация зеркалируется. Если какой-либо узел хранения «сбоит», то всегда найдется другой узел с такой же информацией;
- управляющий узел может быть остановлен без какого-либо влияния на остальные узлы кластера.

При помощи такого подхода к проектированию удалось исключить сбой всей системы из-за сбоя одного узла. Любой узел может быть остановлен без влияния на работу систему в целом.

Рассмотрим этот подход более подробно.

Синхронное зеркалирование. Все данные базы данных зеркалируются в пределах узлов хранения одной группы. Число копий баз данных может варьироваться от 1 до 4. Оно выбирается администратором при запуске кластера. Зеркалирование происходит во время выполнения транзакции. Синхронное зеркалирование позволяет при сбое одного из узлов заменить его менее чем за одну се-

кунду. Если во время выполнения транзакции произошел сбой и не выполнено зеркалирование, приложение уведомляется об этом. Так что всегда есть возможность повторения сбойной транзакции.

Обнаружение сбоя. Различают два способа определения сбойного узла – т.н. «потеря связи» и «потеря пульса». В обоих случаях посылается сообщение всем узлам хранения и определяется дальнейшая возможность функционирования системы.

При сбое возможна ситуация, когда кластер будет поделен на несколько независимо работающих систем. Ее возникновение влечет потерю достоверности и целостности данных в кластере. Для устранения подобных ситуаций используется специальный сетевой протокол разбиений. Согласно ему выбирается одна из работоспособных частей, а остальные узлы перезапускаются и присоединяются к системе как новые.

Потеря связи. Узлы соединяются через различные протоколы. При нормальной работе MySQL Cluster все узлы хранения соединены друг с другом и каждый узел API соединен со всеми узлами хранения. Если узел хранения сообщает, что связь между двумя узлами потеряна, то об этом незамедлительно информируются другие узлы и все вместе они находят сбойный узел. Все сбойные узлы автоматически перезапускаются и подсоединяются к MySQL Cluster. Потеря связи – это наиболее быстрый способ нахождения сбойного узла.

Потеря “пульса”. Существуют сбои узлов, которые невозможно определить при помощи потери связи, такие как проблемы с дисками, оперативной памятью, «истощением» процессора и т.д. Эти сбои влияют на корректную работу узла, но не влияют на связь узла с оставшейся частью кластера. Узлы хранения организованы в логический цикл. Каждый узел посылает сигналы “пульса” следующему узлу хранения в цикле. Узел должен послать последовательно 3 сигнала “пульса” (рис. 2.5). Если он этого не сделал, то следующий в цикле узел хранения классифицирует его как сбойный.

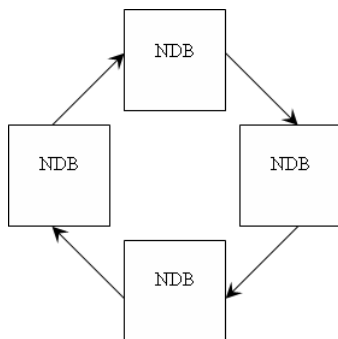


Рис. 2.5

Восстановление системы. В MySQL Cluster используются следующие приемы для восстановления системы после сбоя.

Протоколирование. Во время “нормальных” операций на диск асинхронно пишется протокол операций с базой данных (insert, delete, update и т.д.). Протоколы пишутся на диск каждого узла хранения и используются только при сбое всей системы. Так как протокол содержит все операции изменения, то легко восстановить состояние базы данных на момент сбоя.

Локальные точки восстановления. С ростом числа операций увеличивается и объем протокола. Чтобы он не разрастался, создаются локальные точки восстановления – сохранение базы данных на диск в определенные моменты времени. При сбое отдельного узла хранения восстанавливается сохраненная база, и к ней применяется восстановление по протоколу. Локальные точки восстановления создаются независимо на каждом узле хранения.

Глобальные точки восстановления. Для восстановления после сбоя всей системы создаются глобальные точки восстановления. Одновременно с их формированием происходит формирование локальных точек восстановления на всех узлах хранения.

Процедуры конфигурирования MySQL Cluster

Настройка MGM node выполняется в порядке:

1. Создать каталог /var/lib/mysql-cluster.
2. В каталоге /var/lib/mysql-cluster/ создать файл config.ini:
#секция настроек NDB node по умолчанию
[ndbd default]

#количество копий (1+количество резервных копий)
NoOfReplicas=2
#количество оперативной памяти, отводимое под данные
DataMemory=1600M
#количество оперативной памяти, отводимое под индексы
IndexMemory=250M
#место хранения данных на узлах хранения
DataDir=/var/lib/mysql-cluster/db
#[ndb_mgmd] секция отвечает за параметры MGM node
[ndb_mgmd]
#IP адрес MGM node
HostName=192.168.214.40
[ndb_mgmd]
#IP адрес MGM node
HostName=192.168.214.42
#секция [ndbd] отвечает за параметры NDB node
[ndbd]
#IP адрес NDB node
HostName=192.168.214.41
[ndbd]
#IP адрес NDB node
HostName=192.168.214.45
#секция отвечает за параметры API node
[mysqld]
#IP адрес API node
HostName=192.168.214.40
[mysqld]
#IP адрес SQL node
HostName=192.168.214.42
#настройка стека tcp
[tcp default]
#размер буфера посылаемых данных
SendBufferMemory=2M
#размер буфера принимаемых данных

ReceiveBufferMemory=1M

3. Скопировать из каталога /usr/bin в каталог /var/lib/mysql-cluster файл *ndb_mgmd*.

Настройка NDB node :

1. Создать каталог /var/lib/mysql-cluster.
2. Скопировать из /usr/bin в /var/lib/mysql-cluster файл *ndbd*.

Настройка API node :

1. В каталоге /etc создать файл *my.cnf*.

#настройки, используемые программой mysqlclient

[client]

#порт для подключения к API node

port = 3306

#UNIX сокет для подключения к API node

socket= /tmp/mysql.sock

#настройки для MySQL server

[mysqld]

#порт для подключения к MySQL server

port = 3306

#UNIX сокет для подключения к MySQL server

socket= /tmp/mysql.sock

skip-locking

key_buffer = 256M

max_allowed_packet = 1M

table_cache = 256

sort_buffer_size = 1M

read_buffer_size = 1M

read_rnd_buffer_size = 4M

myisam_sort_buffer_size = 64M

thread_cache_size = 8

query_cache_size= 16M

thread_concurrency = 8

#настройки для использования MySQL Cluster

```

#место хранения схем таблиц
datadir=/var/lib/mysql
#подключение NDB nodes
ndbcluster
#адреса MGM nodes
ndb-connectstring = 192.168.214.40, 192.168.214.42
[mysql-cluster]
Ndbcluster
#адреса MGM nodes
ndb-connectstring = 192.168.214.40, 192.168.214.42
[mysqldump]
quick
max_allowed_packet = 16M
[mysql]
no-auto-rehash
[isamchk]
key_buffer = 128M
sort_buffer_size = 128M
read_buffer = 2M
write_buffer = 2M
[myisamchk]
key_buffer = 128M
sort_buffer_size = 128M
read_buffer = 2M
write_buffer = 2M
[mysqlhotcopy]
interactive-timeout

```

2. Создать каталог /var/lib/mysql. В нем будут храниться системные базы данных и схемы баз данных, используемых MySQL Cluster.

3. Создать группу mysql:

```
shell> groupadd mysql
```

4. Создать пользователя mysql:
shell>useradd -g mysql mysql
5. Перейти в каталог /usr/local/mysql:
shell>cd /usr/local/mysql
6. Изменить владельца каталога:
shell> chown -R mysql
7. Изменить группу-владельца каталога:
shell> chgrp -R mysql
8. Создать системные таблицы SQL node:
/usr/bin/mysql_install_db --user=mysql
9. Изменить владельца каталога:
shell> chown -R root
10. Изменить владельца каталога данных:
shell> chown -R mysql <каталог_баз_данных>

Управление MySQL Cluster

Консоль управления ndb_mgm. В комплект поставки MySQL Cluster входит бинарный файл *ndb_mgm*. После компиляции он находится в каталоге /usr/local/bin. Запуск производится командой /usr/local/bin/ndb_mgm. Файл может быть запущен на любом узле MySQL Cluster. Ниже приведено описание команд, используемых для управления кластером:

1. HELP – выводит список доступных команд.
2. SHOW – выводит состояние MySQL Cluster.
3. CONNECT ip_адрес_MGM_node – подключение к MGM node.
4. номер_узла START – запуск узла хранения с номером номер_узла. Для работы команды необходимо запустить узел хранения с опцией -n или -nostart.
5. номер_узла STOP – останов узла хранения с номером номер_узла.
6. номер_узла RESTART – перезапуск узла хранения с номером номер_узла.

7. номер_узла STATUS – вывод информацию о состоянии узла.

8. ENTER SINGLE USER MODE номер_узла – однопользовательский режим, в котором доступ к MySQL Cluster имеет доступ только SQL node с номером номер_узла.

9. EXIT SINGLE USER MODE – выход из однопользовательского режима.

10. SHUTDOWN – останов кластера, за исключением SQL nodes.

11. QUIT, EXIT – выход из управляющей консоли.

Порядок запуска MySQL Cluster:

1. На каждом MGM node выполнить команду:

```
shell>/var/lib/mysql-cluster/ndb_mgmd --config-file=/etc/config.ini
```

2. На каждом NDB node выполнить команду:

```
shell>/var/lib/mysql-cluster/ndbd --connect-string = "<список_узлов_управления>" [--initial]
```

где <список_узлов_управления> – IP адреса MGM nodes, написанные через запятую. Если требуется очистить память NDB node от данных, то необходимо добавить ключ --initial.

3. На каждом SQL node выполнить команду:

```
shell>/usr/bin/mysqld_safe &
```

Порядок останова кластера. Для штатного останова используется управляющая консоль:

```
ndb_mgm>connect <ip_адрес_узла_управления>
```

```
ndb_mgm>show
```

```
ndb_mgm>shutdown
```

SQL node останавливаются при необходимости вручную:

```
shell>ps ax | grep mysql
```

Будет выведен список процессов. Первым необходимо останавливать процесс, содержащий *mysqld_safe*. Вторым – процесс *mysqld*.

Практика 5. Лабораторная работа

1. Запустить MySQL Clustert в слеующих конфигурациях (IP адреса узлов указывает преподаватель):

а) 1 MGM + 1 API + 2 NDB

б) 1 MGM + 2 API + 2 NDB

в) 1 MGM + 3 API + 2 NDB

2. Запустить на всех 3 конфигурациях запросы к базе данных и зафиксировать время выполнения.

Для автоматического тестирования одной из конфигураций MySQL Cluster необходимо выполнить скрипт test, находящийся в каталоге /usr/sbin, следующим образом:

```
shell>/usr/sbin/test <номер_конфигурации>
```

где <номер_конфигурации> = 0,1,2.

Для ручного тестирования MySQL Cluster необходимо соединиться с одним из его API узлов:

```
shell>mysql -h ip_адрес_API [-u имя_пользователя] [-ппароль]
```

3. Подсчитать коэффициент ускорения относительно первой конфигурации.

Занятие 6.

ПАРАЛЛЕЛЬНАЯ СУБД Clusterix

Целью этого занятия является ознакомление с принципами построения параллельной СУБД Clusterix, получение навыков работы на кластере баз данных.

Лекция 6. Знакомство с принципами построения [9]

Общее описание. Рассматриваемая СУБД ориентирована на использование стандартного общедоступного аппаратно-программного обеспечения (Beowulf-технология). Аппаратно СУБД Clusterix функционирует на кластере из персональных компьютерных (ПК), соединенных локальной сетью FastEthernet или GigabitEthernet. Программная среда параллельной СУБД Clusterix представляет собой “надстройку” над серверами СУБД MySQL, установленными на всех узлах кластера. В функции СУБД MySQL входит реализа-

ция таких низкоуровневых операций с БД, как хранение и обработка отношений БД в виде специализированных файлов.

Функции программной надстройки заключаются:

- в организации параллельной обработки запроса пользователя различными узлами кластера;
- в управлении и мониторинге за состоянием узлов кластера;
- в преобразовании исходного SQL-запроса пользователя в соответствующий план обработки, который обеспечивает параллельное его исполнение.

Взаимодействие между надстройкой и серверами СУБД MySQL осуществляется передачей SQL-запросов (команды) и через файловую подсистему (данные).

Программное обеспечение кластера функционирует под операционной системой Linux.

В основу параллельной обработки положена идея конвейерной обработки запроса [10]. Эта идея реализуется построением специального план обработки запроса (рис 2.6) по схеме:

“SELECT – PROJECT - JOIN”.

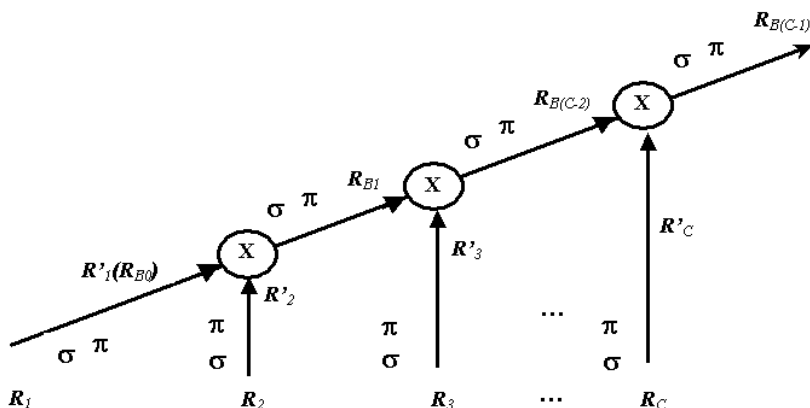


Рис. 2.6

На рис.2.6: R_i – одно из отношений, участвующих в обработке запроса, $i=1 \dots C$. Здесь C – количество отношений, задействованных в запросе; R'_i – промежуточное отношение, $i=1 \dots C$; R_{Bj} – временное отношение, $j=1 \dots C - 1$; σ – операция селекции; π – операция проекции; x – операция декартова произведения.

Обработка запроса по указанной схеме проводится в 3 этапа.

На первом этапе выполняется выборка кортежей из исходного отношения БД (R_i), отфильтрованных условием *селекции* (σ). На втором этапе над этими кортежами выполняется операция *проекции* (π) только тех полей (атрибутов) отношения, которые будут участвовать в последующей обработке или должны присутствовать в конечном результате обработки запроса. Назначением операций σ и π является уменьшение размеров исходного отношения БД перед третьим этапом – операцией *соединения*. Выполнение операции соединения является одной из наиболее трудоемких операций. Ее эффективность напрямую зависит от размеров отношений, участвующих в соединении.

Система предусматривает двухуровневую обработку данных: верхний и нижний уровни обработки.

Нижний уровень выполняет операции селекции (σ) и проекции (π) над исходными отношениями R_i базы данных. Результатом обработки нижнего уровня является промежуточное отношение R'_i . Далее промежуточное отношение передается на верхний уровень для дальнейшей обработки – операции соединения.

На *верхнем уровне* выполняется операция соединения между отношениями R'_i и $R_{B(i-2)}$. Результатом выполнения соединения на i -шаге является временное отношение $R_{B(i-1)}$.

Операция соединения

$$R \blacktriangleright \blacktriangleleft S = \pi(\sigma_\theta(R \times S)).$$

выражается через операции селекции, проекции и декартова произведения.

Отношения БД распределены по дискам на процессорах нижнего уровня (IO_r). Распределение отношений осуществляется горизонтально с применением хеш-функции к первичному ключу для каждого кортежа отношения.

Процессоры верхнего уровня обработки запроса называются процессорами JOIN. Количество процессоров JOIN равно количеству процессоров IO .

Кроме исполнительных процессоров (IO и $JOIN$), есть еще два процессора. Процессор MONITOR реализует функции мониторинга и управления остальными процессорами системы. Процессор (препроцессор) MTRANS предназначен для претрансляции исход-

ного запроса пользователя к виду регулярного дерева обработки запроса (рис. 2.6). Оба они функционируют на *Host* ЭВМ.

Принятый план обработки ориентирован на реализацию следующих принципов:

- параллелизм;
- конвейерность;
- регулярность процедуры сборки промежуточных отношений для выполнения страничного соединения;
- реализация промежуточного хеширования для повышения эффективности страничных соединений и уменьшения объемов сетевых передач.

Оператор **SELECT** осуществляет выборку кортежей из отношений БД, **PROJECT** – проекцию полей отношения, **JOIN** выполняет соединение. Операторы **SELECT** и **PROJECT** относятся к первому, **JOIN** – ко второму уровню обработки данных в кластере. Операторы первого уровня выделены в отдельный программный модуль – процессор ввода/вывода (**IO**), операции второго уровня – в процессор **JOIN**.

Основные программные модули («логические» процессоры) разрабатываемой системы:

- Модуль мониторинга и управления **mlisten** (**MONITOR**) – получение SQL запросов пользователя, передача результатов обработки, управление компонентами системы.
- Модуль ввода/вывода **irun** (**IO**) – выполнение операций селекции, проекции, ввода/вывода.
- Модуль соединения **jrun** (**JOIN**) – выполнение операций сортировки, соединения, проекции над промежуточными результатами обработки.
- Модуль сортировки и агрегации **msort** (**SORT**) – выполнение операций агрегации, группировки и сортировки над результатами обработки.
- Модуль трансляции **mtrans** (**MTRANS**) – выполнение трансляции SQL запросов во внутреннее представление системы.

На каждом физическом процессоре любого уровня могут быть запущены несколько «логических». Число «логических» про-

цессоров на обоих уровнях всегда одинаково, а отношение $k = q/p$ необходимо целое, $k = 1, 2, \dots$, где q – число физических процессоров JOIN, p – число физических процессоров IO.

Благодаря модульному построению можно получить несколько вариантов архитектуры системы: «линейка» (рис.2.7), «симметрия» (рис.2.8), «асимметрия» (рис.2.9). Здесь кружками обозначены «логические» процессоры, прямоугольниками – физические процессоры.

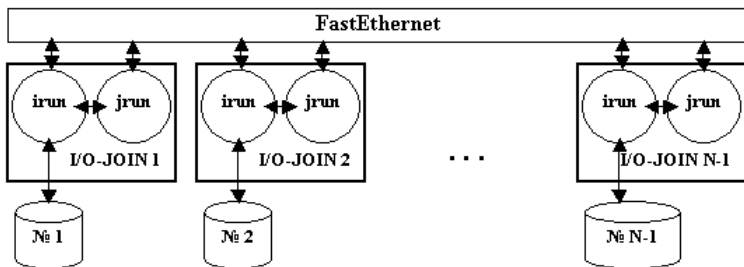


Рис. 2.7

Архитектура «линейка» имеет место при совмещении на одном физическом процессоре двух «логических» – IO и JOIN ($k \rightarrow \infty$).

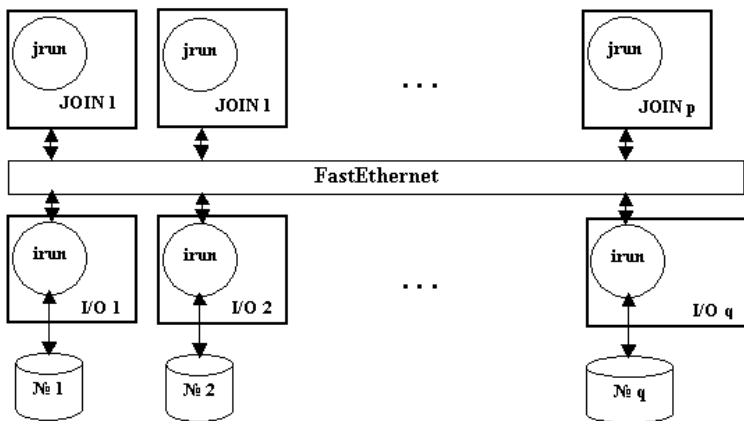


Рис.2.8

Архитектура «симметрия» получается в том случае, если число процессоров верхнего уровня равно числу процессоров нижнего уровня. При этом на каждый «логический» процессор выделяется один физический процессор ($k = 1$).

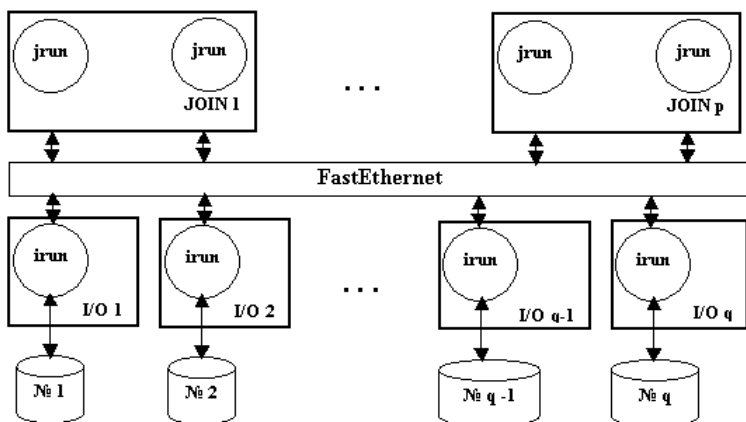


Рис.2.9

Архитектура «асимметрия» имеет место, если число процессоров верхнего уровня (JOIN) не равно числу процессоров нижнего уровня (I/O). Обычно здесь $k = 2, 3, \dots$, т.е. на одном «физическом» процессоре JOIN совмещаются два и более «логических».

Взаимодействие между модулями осуществляется передачей сообщений сетевыми протоколами TCP/IP и реализовано через «сокеты». Программные модули написаны с использованием технологий параллельного программирования (threads, mutex, shared memory). На каждом процессоре функционирует СУБД MySQL, которая выполняет низкоуровневые операции: работа с файлами, индексами и т.д.

Алгоритм работы системы. Точкой входа в систему для пользователя является Host-машина. Пользователь передает свой SQL-запрос с помощью программы-клиента *sql_client*. Программа *sql_client* подключается к модулю *mlisten* на заранее известный сетевой порт и посылает SQL-запрос в виде строки. Модуль *mlisten* при подключении данного пользователя запускает для его обработки функцию-поток *Процесс h*, где h – порядковый номер запроса в системе. Получив строку запроса, *Процесс h* передает ее в подсистему трансляции, которая преобразует исходный запрос к плану обработки запроса.

План обработки данного запроса формируется в виде набора SQL-команд для каждого из трех программных модулей: *irun*, *jrun* и

msort. Команды пересылаются соответствующим модулям, а сам *mlisten* переходит в режим ожидания результата обработки. На каждый запрос пользователя выделяется отдельный поток-обработчик (функция *Client* файла *mlisten.cpp*). Первым к обработке запросов приступает модуль *irun*. На рис. 2.9 поток управляющих команд показан пунктиром, поток данных – сплошной линией, сетевые интерфейсы – черными точками.

Модуль *irun* выполняет несколько потоков-обработчиков. Каждый обработчик функционально ориентирован. Получив пакет команд от Процесс *h*, модуль *irun* помещает его во входную очередь команд (обработчик *ServGet*). Другой обработчик – *Run* – занимается тем, что последовательно выбирает команды из входной очереди команд и с помощью API-функций отправляет их на выполнение СУБД MySQL. От СУБД результат обработки возвращается в виде буфера строк, каждая строка которого есть кортеж результата. Далее над этим буфером проводится операция динамического сегментирования по другим модулям *irun*.

В результате на каждом модуле *irun* формируется промежуточное отношение в виде трех файлов формата MySQL (*.frm – файл структуры отношения, *.MYD – файл хранения данных отношения, *.MYI – файл хранения индексов отношения). Полученное отношение передается соответствующему модулю *jrun*, а сам модуль *irun* приступает к выполнению следующего SQL-запроса из входного буфера команд. Это происходит до полного освобождения входного буфера.

Модуль *jrun* помещает полученный от *mlisten* пакет команд во входную очередь. После приема (обработчик *ServGet*) промежуточного отношения от *irun* модуль *jrun* помещает его в системный каталог СУБД MySQL (по умолчанию */var/lib/mysql/in_*). Из входного буфера извлекается текущая команда, которая передается на выполнение СУБД MySQL.

Команда состоит из трех операций. Первая операция создает индекс временного отношения по атрибуту (обработчик *Index*), который будет принимать участие в операции соединения на текущем шаге. Вторая – непосредственно операция соединения. Ее результатом является буфер строк, который затем сегментируется по другим модулям *jrun*. Операция динамического сегментирования осуществляется на основе атрибута, по которому будет производиться

соединение на следующем шаге. В результате этих двух операций формируется временное отношение.

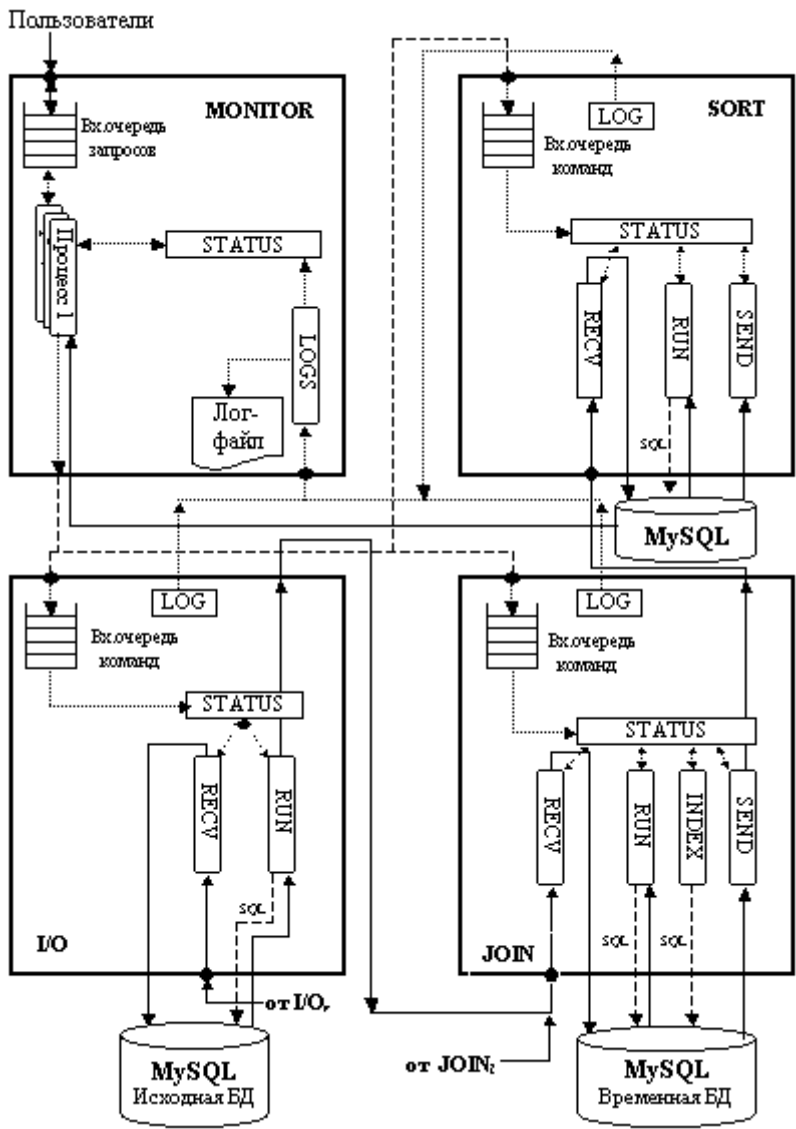


Рис. 2.9

И, наконец, последняя операция – создание индекса временного отношения по атрибуту, по которому проводилось сегментирование.

При достижении конца очереди команд для текущего запроса полученное временное отношение пересылается на следующий уровень обработки – модуль *msort*. Сам же модуль *jrun* приступает к обработке очередного запроса при наличии соответствующего временного отношения. Здесь иницилирующим действием на запуск алгоритма обработки является получение промежуточного отношения от *irun*. Тем самым реализуется управление по готовности данных.

Модуль **msort** также имеет входную очередь команд. Каждый запрос состоит из одной команды. При получении временных отношений от всех модулей *jrun* модуль *msort* объединяет их содержимое в одно результирующее отношение. Далее к отношению применяется операция агрегации и сортировки, если они предусмотрены в исходном запросе пользователя. Модуль *mlisten* определяет факт завершения обработки запроса модулем *msort* и выходит из цикла ожидания. Полученное отношение передается пользователю в качестве результата.

На этом процесс обработки текущего запроса завершается.

Каждый модуль имеет в своем составе структуру STATUS (рис.2.9). Эта структура выполняет несколько функций:

- сбор и хранение информации о прохождении запросом различных стадий обработки в рамках модуля;
- организация взаимодействия (локальной маршрутизации) между одновременно работающими обработчиками;
- обеспечение строгой последовательности обработки запроса.

Наличие двух уровней обработки с использованием СУБД MySQL обуславливает необходимость перемещения данных между уровнями обработки (IO, JOIN, SORT). Кроме хеширования отношений при распределении на страницы, в системе реализован механизм динамического сегментирования временных и промежуточных отношений на основе хеширования по полям, участвующим в операции соединения.

В системе реализован алгоритм операции соединения по равенству полей. Он не является универсальным. Поэтому для всех

других вариантов соединения приходится организовывать “кольцо”, когда промежуточное отношение от модуля IO последовательно передается через все модули JOIN.

Хранение данных в системе. Данные СУБД *горизонтально* распределены по узлам IO. Распределение данных между узлами IO осуществляется с помощью хеширования значений первичного ключа. Для отношения, первичный ключ которого состоит из нескольких полей, функция хеширования

$$\text{hash} = (\text{key_field1} \bmod M + \dots + \text{keyfieldP} \bmod M) \bmod M,$$

где P – количество полей в первичном ключе; основание деления по модулю, \bmod – операция деления по модулю. Значение *hash* показывает порядковый номер процессора IO, на который распределяется текущий кортеж.

Команды управления кластером. Для управления кластером используется программа-скрипт *mgm_clusterix*:

```
shell> mgm_clusterix имя_команды.
```

Основные функции скрипта:

- задание архитектуры и конфигурации кластера (команда *set_conf*);
- запуск (команда *start*) и останов (команда *stop*) кластера;
- формирование тестовой базы данных (команда *db*);
- проверка статуса кластера (команда *stat*);
- обновление версии программного обеспечения (команда *conf*);
- перезагрузка узлов кластера (команда *reboot*);
- выключение узлов кластера (команда *poweroff*);
- вызов справки по всем функциям системы (команда *help*).

Удаленное управление узлами кластера осуществляется с помощью службы *ssh*. Для этого на каждый узел установлен агент *ssh*, сформированы и распределены ключи. Прежде чем начать выполнение команд с помощью *mgm_clusterix*, необходимо выполнить следующие команды:

```
shell> ssh-agent $SHELL
```

```
shell> ssh-add
```

После этого система попросит ввести пароль для ключа. Операционная система будет хранить пароли в специальном сис-

темном буфере и подставлять их по мере необходимости. Если этого не сделать, система будет запрашивать пароль каждый раз при выполнении команды *ssh* на удаленном узле кластера.

Запуск системы осуществляется командой:

```
shell> ./mgm_clusterix start
```

Результатом ее выполнения является список стартовавших модулей с их IP – адресами и номерами сетевых портов.

Останов системы реализуется командой:

```
shell> ./mgm_clusterix stop
```

Результатом выполнения является список остановленных модулей с их IP – адресами и номерами сетевых портов.

Задание конфигурации. Конфигурация системы задается командой:

```
shell> ./mgm_clusterix set_conf номер_конфигурации
```

Параметр *номер_конфигурации* представляет собой условное обозначение конфигурации системы. В табл.2.1 представлены возможные значения этого параметра.

Таблица 2.1

Номер конфигурации	Архитектура	Кол-во модулей I/O	Кол-во модулей JOIN	Кол-во узлов кластера
422	симметрия	2	2	4
433	симметрия	3	3	6
444	симметрия	4	4	8
522	линейка	2	2	2
544	линейка	4	4	4
588	линейка	8	8	8
424	асимметрия	4	2	6
426	асимметрия	6	2	8

Проверка статуса системы осуществляется командой

```
shell> ./mgm_clusterix stat
```

Результатом работы команды является список модулей с их IP-адресами, номерами сетевых портов и статусом выполнения («Running»– выполняется, «Stoped» – остановлен, «Not Found» – не определен).

Формирование тестовой базы данных. В качестве тестовой базы данных используется база данных теста TPC-D. Для каждой конфигурации формируется своя тестовая база данных. Ее формирование осуществляется командой

```
shell> ./mgm_clusterix db
```

Эта команда включает в себя следующие этапы выполнения:

- хеширование отношений исходной базы данных по первичным ключам для каждого IO-узла;
- формирование фрагментированных отношений тестовой базы данных;
- распределение фрагментированных отношений тестовой базы данных по IO-узлам системы.

Обновление версии программного обеспечения. Эта функция реализуется командой:

```
shell> ./mgm_clusterix conf.
```

Команда *conf* состоит из следующих операций:

- замена программных модулей на удаленных узлах кластера версией с сервера управления;
- удаление всех отношений из временной базы данных на удаленных узлах кластера.

Перезагрузка узлов кластера реализуется командой:

```
shell> ./mgm_clusterix reboot.
```

Результатом выполнения этой команды является перезагрузка операционной системы на всех узлах кластера.

Выключение узлов кластера реализуется командой:

```
shell> ./mgm_clusterix poweroff
```

Результатом выполнения этой команды является выключение питания узлов кластера.

Запуск кластера выполняется в следующем порядке:

1. задать текущую конфигурацию кластера;
2. распределить базу данных по множеству узлов кластера;
3. запустить кластер;
4. проверить текущее состояние кластера;
5. передать на выполнение SQL – запрос, зафиксировать результат обработки запроса;

6. остановить кластер.

Соответствующая последовательность команд:

- задание конфигурации кластера
shell> mgm_clusterix set_conf номер_конфигурации
- подготовка БД для текущей конфигурации
shell> mgm_clusterix db
- запуск кластера
shell> mgm_clusterix start
- проверка статуса кластера
shell> mgm_clusterix stat
- запуск SQL-запроса
shell> sql_client IP_адрес Port_номер SQL_команда
- остановка работы кластера
shell> mgm_clusterix stop

Практика 6. Лабораторная работа

1. Выполнить запуск кластера БД на различных конфигурациях:

- конфигурация «симметрия» для 4 узлов IO и 4 узлов JOIN;
- конфигурация «линейка» для 8 узлов кластера.

2. Запустить на одной из конфигураций кластера запросы к базе данных.

3. Запустить одинаковую серию из 3-х запросов на «линейке» при числе узлов кластера $N = 2, 4, 8$. Зафиксировать в отчете время выполнения серии запросов для каждой конфигурации. Подсчитать коэффициент ускорения относительно $N = 2$.

4. Запустить одинаковую серию из 3 запросов на различных конфигурациях кластера. Зафиксировать в отчете времена выполнения каждого запроса и среднее время выполнения серии запросов для каждой конфигурации.

Занятие 7.

ОБРАБОТКА ЗАПРОСОВ В СУБД CLUSTERIX

Назначение СУБД Clusterix – параллельное выполнение SQL-запросов за минимально возможное время. Для достижения этой цели исходный запрос пользователя преобразуется к виду, позволяющему выполнять части исходного запроса параллельно. Наиболее трудоемкими операциями СУБД являются сортировка и соединение.

Лекция 7. Претрансляция и исполнение запросов [9, 11]

Формирование команд плана обработки запросов

Пример построения дерева обработки. Рассмотрим конфигурацию, содержащую 2 узла IO и 2 узла JOIN. Пусть имеется база данных с тремя отношениями A, B, C:

Отношение A: A1* A2 A3

Отношение B: B1* B2

Отношение C: C1* C2* C3

* – атрибуты, входящие в первичный ключ отношения.

Построим дерево обработки запроса вида рис.2.6 для следующего запроса:

```
SELECT A2, SUM(C3) FROM A, B, C
WHERE A1=C2 AND B1=C1 AND B2>50
GROUP BY A2
```

Для упрощения процесса построения дерева на начальных стадиях будем считать, что 1) имеется по одному узлу на каждый логический процессор обработки (IO, JOIN, SORT); 2) одни и те же команды выполняются параллельно на нескольких узлах обработки (IO, JOIN).

Сначала надо определить, сколько отношений присутствует в запросе. По части запроса за ключевым словом FROM убеждаемся в том, что в нашем случае таких отношений три – A, B, C. Поэтому схема плана обработки отвечает рис. 2.10.

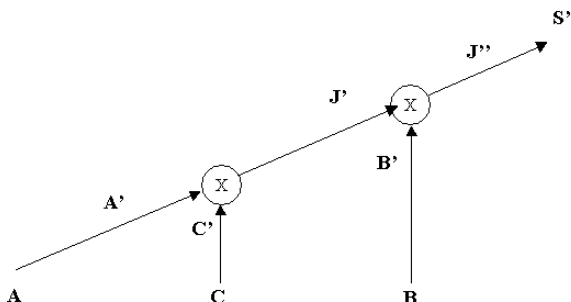


Рис. 2.10

Итак, общая схема обработки запроса известна. Остается определить конкретные команды этого плана.

Состав команд. Все планы обработки запроса Clusterix состоят из трех типов команд: для IO, для JOIN, для SORT. Формирование команд идет слева направо, по ходу выполнения запроса.

Так как обработка запроса начинается с IO, определим первую команду для IO – A`. В результате выполнения этой команды получается промежуточное отношение, содержащее все атрибуты отношения A, необходимые для дальнейшей обработки запроса.

Общий вид команды для IO выглядит следующим образом:

```
SELECT атрибуты_отношения FROM имя_отношения WHERE
условие_селекции
```

Сначала определяется имя текущего отношения. В нашем случае – это отношение A.

Затем определяются *атрибуты_отношения*. Если атрибут присутствует в отношении A, но отсутствует в данном запросе, то его использовать не нужно. В этом состоит смысл операции *проекция* – исключение из обработки неиспользуемых атрибутов. Перечислим все атрибуты отношения A, участвующие в рассматриваемом запросе. Таких атрибутов два: A1 и A2. Атрибут A3 имеется в исходном отношении A, но в данном запросе не используется. Поэтому он не включается в команду.

Далее определяется *условие_селекции*. Оно задается условиями исходного запроса

```
<атрибут_отношения_A> условие <константа>,
```

которые располагаются после ключевого слова WHERE. Для отношения A таких условий в рассматриваемом запросе нет. Но есть условие для отношения B ($B2 > 50$).

Поэтому первая команда для IO – A`:

```
SELECT A1, A2 FROM A.
```

Аналогично предыдущему, выделим атрибуты отношения C. В рассматриваемом примере – это C1, C2, C3. Условия селекции отсутствуют.

Соответственно вторая команда – C`:

```
SELECT C1, C2, C3 FROM C.
```

Таким образом, сформированы два промежуточных отношения – A` и C`, которые соединяются условием $A1 = C2$.

Сформируем первую команду для JOIN – J`.

Общий вид команд соединения:

```
SELECT атрибуты_отношения FROM имя_отношения1,  
      имя_отношения2 WHERE условие_соединения.
```

Условие_соединения известно ($A1 = C2$). Имя_отношения1 – A`, имя_отношения2 – C`.

Определим список атрибутов, используемых на более поздних стадиях обработки запроса. Оставшиеся еще не рассмотренными условия позволяют заключить, что из всех атрибутов отношений A` и C` в дальнейшей обработке принимают участие A2 (входит в конечный результат), C1 (участвует в операции соединения), C3 (входит в конечный результат).

Итак, команда J`:

```
SELECT A2, C1, C3 FROM A`, C` WHERE A1=C2
```

Следующей будет команда B`. Поступаем тем же способом, что и для A` и C`. Но не забудем, что в исходном запросе имеется условие селекции $B2 > 50$, которое нужно включить в команду.

Таким образом, команда B`:

```
SELECT B1 FROM B WHERE B2>50.
```

Атрибут B2 используется только в условии селекции и в дальнейшем не используется.

Очередная команда – это соединение J'' отношения B' и результата предыдущего соединения по условию $B1=C1$. Определим список атрибутов отношения J'' . На заключительном этапе обработки потребуются только два атрибута $A2$ и $C3$. Оба входят в результат.

Команда J'' :

SELECT $A2, C3$ FROM J', B' WHERE $B1=C1$.

В исходном запросе имеется операция агрегации **SUM**, которая выполняется на «логическом» процессоре **SORT**.

Общий вид команд для **SORT**:

**SELECT атрибуты FROM имя_отношения
GROUP BY атрибуты_агрегации
ORDER BY атрибуты_сортировки.**

Эта команда в большинстве случаев повторяет исходный запрос.

Команда S' :

SELECT $A2, SUM(C3)$ FROM J'' GROUP BY $A2$.

В конечном итоге имеем:

Команды плана обработки запроса для узлов IO

A' : SELECT $A1\%2, A1, A2$ FROM A

C' : SELECT $C2\%2, C1, C2, C3$ FROM C

B' : SELECT $B1\%2, B1$ FROM B WHERE $B2>50$

Команды плана запроса для узлов JOIN

J' : SELECT $C1\%2, A2, C1, C3$ FROM A', C' WHERE $A1=C2$

J'' : SELECT $A2\%2, A2, C3$ FROM J', B' WHERE $B1=C1$

Команды плана запроса для SORT:

S' : SELECT $A2, SUM(C3)$ FROM J'' GROUP BY $A2$

Параллельная обработка запроса

Сформированные команды выполняются параллельно на соответствующих узлах кластера. Как уже отмечалось выше, отношения базы данных горизонтально распределены по узлам IO. При этом каждый узел независимо обрабатывает собственную часть данных.

Параллельный алгоритм соединения. Для выполнения операции соединения в системе применен параллельный алгоритм с использованием хеширования [12]. Хеширование в данной системе используется в двух разных случаях: 1) для распределения исходного отношения по множеству узлов хранения IO; 2) при реализации операции соединения.

Применение хеширования перед непосредственным *соединением* позволяет значительно сократить время выполнения этой операции. В процессе хеширования оба соединяемых отношения разделяются на непересекающиеся блоки отношений меньшего размера, над которыми может независимо (параллельно) выполняться операция соединения. При этом для каждой пары таких блоков выделяется отдельный процессор JOIN.

В данном случае функция хеширования применяется к атрибутам отношений, по которым выполняется соединение. Данный алгоритм рассчитан только на один тип операции соединения – *соединения по условию равенства атрибутов*.

Выполнение сформированного плана обработки по тактам. На следующих рисунках представлена реализация параллельного алгоритма обработки запроса для рассматриваемого примера.

Такт №1 (*выполнение команды A' на IO – рис. 2.15*).

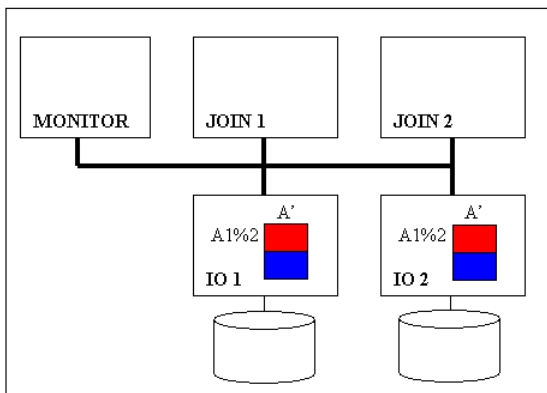


Рис. 2.15

Выполнение запроса начинается с команды: A' :

$SELECT A1\%2, A1, A2 FROM A.$

Выражение $A1\%2$ реализует операцию хеширования, о которой было сказано выше. В качестве функции хеширования использован остаток от деления по модулю 2. Значение 2 в функции хеширования (2 узла JOIN) показывает, на сколько блоков разбивается результат выполнения команды A' перед операцией соединения. Если бы текущая конфигурация кластера содержала 4 узла JOIN, то функция хеширования выглядела бы как $A1\%4$.

В результате выполнения команды A' каждая запись будет иметь признак, принимающий значение 0 либо 1. Этот признак указывает номер узла JOIN, на который должна быть переправлена данная запись.

Такт №2 (*динамическая сегментация на IO – по атрибуту $A1$*). Перед отправкой результата выполнения команды A' на отдельные узлы JOIN выполняется сборка записей с одинаковым значением признака на соответствующие узлы IO (рис.2.16).

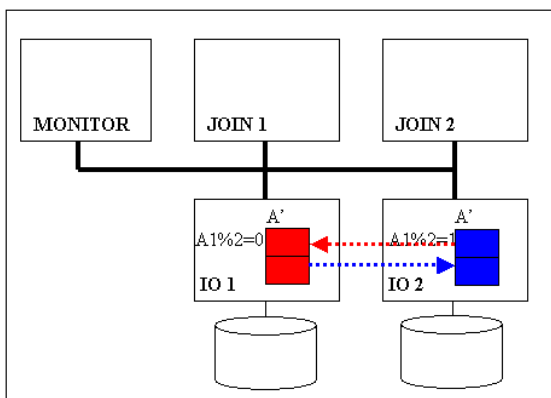


Рис. 2.16

Такт №3 (*пересылка первого промежуточного отношения с IO на JOIN*). Полученные ранее блоки первого промежуточного отношения пересылаются узлам JOIN (рис.2.17).

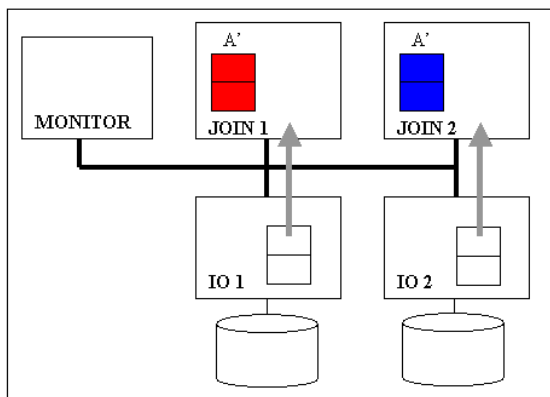


Рис. 2.17

Такт №4 (*выполнение второй команды на IO*). После передачи результатов команды A' узлы IO выполняют следующую команду плана обработки (команду B') (рис.2.18). Узлы JOIN ожидают результаты этого действия, чтобы начать выполнение соединения.

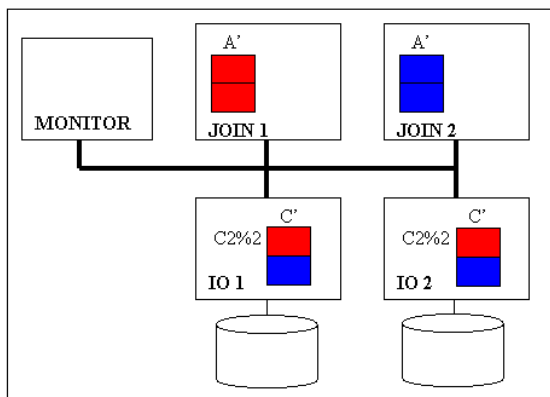


Рис. 2.18

Такт №5 (*динамическая сегментация на IO – по атрибуту C2*). Если результаты команды A' хешировались по атрибуту $A1$, то результат команды C' хешируется по атрибуту $C2$ (рис. 2.19). Именно по этим атрибутам ($A1$ и $C2$) будет выполняться операция соединения.

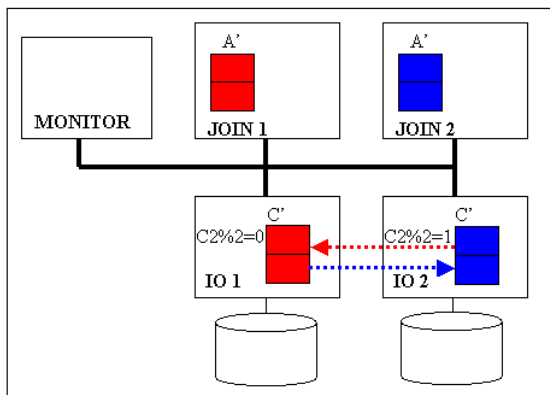


Рис. 2.19

Такт №6 (пересылка второго промежуточного отношения на *JOIN*). На узлы *JOIN* пересылаются результаты выполнения команды *C'* (рис. 2.20).

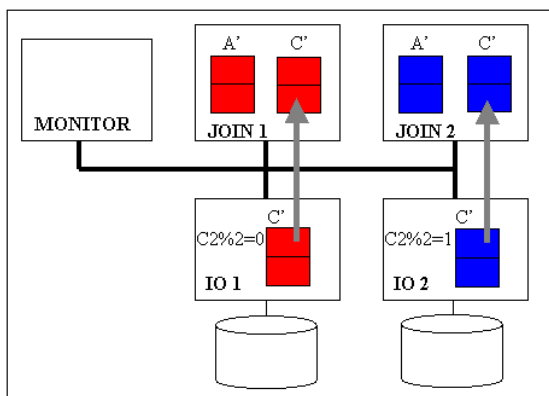


Рис. 2.20

Такт №7 (Параллельное выполнение команд на *IO* и *JOIN* – внутрizaпросный параллелизм). На данном такте параллельно выполняется очередной запрос на узлах *IO* и операция соединения на узлах *JOIN* (рис. 2.21).

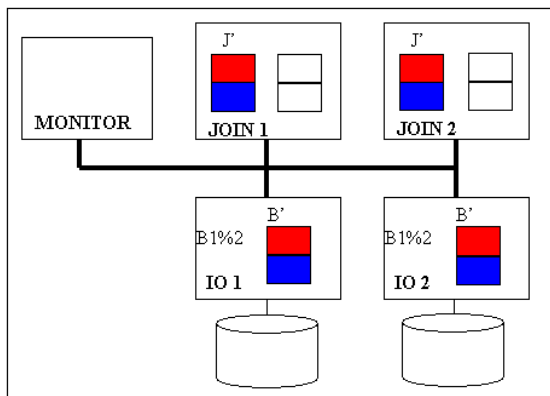


Рис. 2.21

Такт №8 (*динамическая сегментация: на IO – по атрибуту B1; на JOIN – по атрибуту C1*). Как видно из рис. 2.22, операция хеширования выполняется не только на IO, но и на JOIN. И там, и там происходит сборка записей, имеющих одинаковое значение функции хеширования .

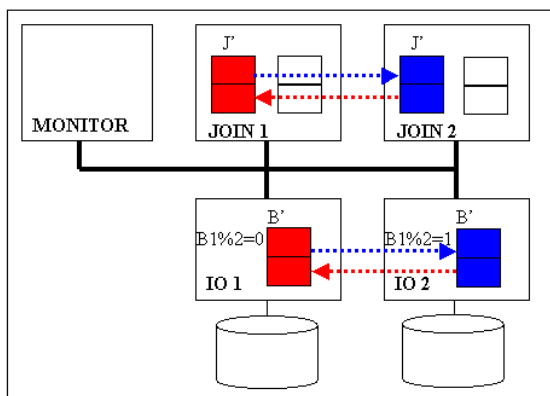


Рис. 2.22

Такт №9 (*пересылка третьего промежуточного отношения с IO на JOIN*). В этом такте на JOIN пересылается результат выполнения команды B' (рис. 2.23).

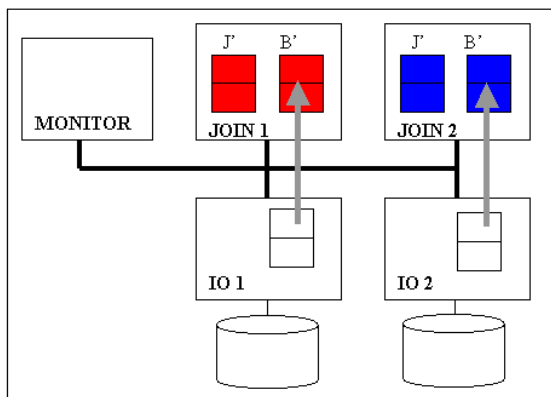


Рис. 2.23

Такт №10 (выполнение второй команды на JOIN). Узлы JOIN приступают к выполнению команды J'', а узлы IO могут приступать к выполнению первой команды следующего запроса пользователя (помечен менее интенсивным цветом) (рис.2.24).

Такт №11 (выполнение команды на SORT). После выполнения последней операции соединения его результат пересылается в модуль SORT (находится на узле MONITOR) (рис. 2.25). Этот модуль сначала выполняет конкатенацию результатов с узлов JOIN, а затем – агрегацию и сортировку результата.

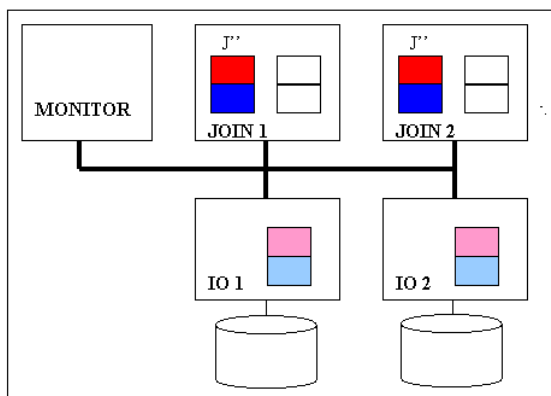


Рис. 2.24

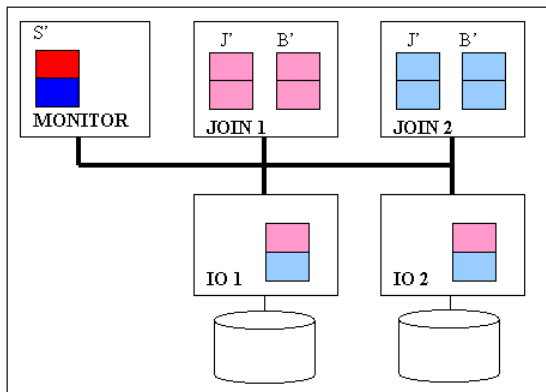


Рис. 2.25

На этом обработка запроса заканчивается, результат обработки передается пользователю.

Практика 7. Лабораторная работа

1. Получить у преподавателя пример SQL-запроса.
2. Составить для этого запроса схему обработки в виде соответствующего плана.
3. Написать команды для этого плана.

ЛИТЕРАТУРА К РАЗДЕЛУ II

1. Озкарахан Э. Машины баз данных и управление базами данных: Пер. с англ. – М.: Мир, 1989.
2. Ульман Дж. Основы систем баз данных – М.: Финансы и статистика, 1983.
3. Калиниченко Л.А., Рывкин В.М. Машины баз данных и знаний. – М.: Наука, 1990.
4. Stonebraker M. The case for shared nothing // Database Engineering Bulletin. March 1986. Vol. 9. No 1. P.4-9.
5. Соколинский Л.Б. Обзор архитектур параллельных систем баз данных // Программирование. 2004. №6. С.1-15
6. Оззу М.Т., Валдуриз П. Распределенные и параллельные системы баз данных // СУБД. 1996. №4.

7. DeWitt D.J., Gray J. Parallel Database Systems: The future of high – performance database systems //Communications of the ACM. 1992 V.35. № 6. P.85-98.
8. <http://www.mysql.com>
9. Абрамов Е.В. Параллельная СУБД Clusterix. Разработка прототипа и его натурное исследование //Вестник КГТУ им. А.Н. Туполева. 2006. №2. С.52-55.
10. Райхлин В.А. Моделирование машин баз данных распределенной архитектуры // Программирование. 1996. №2. С.7-16.
11. Абрамов Е.В., Куревин В.В. Разработка и реализация претрансляции запросов для РС-кластеров баз данных //XIV Туполевские чтения. Материалы конференции. Том IV. Казань, 2006. С.39-40.
12. Kuitsuregawa M., Ogawa Y. A New Parallel Has Join Method with Robustness for Data Skew in Super Database Computer (SDC), Proceedings of the Sixteenth International Conference on Very Large Data Bases, Melbourne, Australia, August, 1990. P.210-221

III. ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ЗАЩИЩЕННЫХ КАРТОГРАФИЧЕСКИХ БАЗ ДАННЫХ

Защита данных в геоинформатике актуальна вследствие наличия на картах различных конфиденциальных сведений: стратегического характера, о местах локализации редких животных и растений, особо ценных полезных ископаемых и др. Стратегия и тактика защиты данных геоинформационных систем (ГИС) требуют серьезного внимания. Этот раздел знакомит с двумерно-ассоциативным механизмом защиты картографических данных и вопросами построения защищенной картографической базы данных, с ГИС MapInfo Professional и кластерной крипто-картографической системой управления базами данных Security Map Cluster.

Введение

НЕОБХОДИМЫЕ СВЕДЕНИЯ ИЗ КАРТОГРАФИИ

В картографии приняты: географическая (геодезическая), прямоугольная и полярная системы координат [1,2].

Географические координаты представляют собой угловые величины – широту и долготу, которые определяют положение точек на земной поверхности относительно экватора и меридиана, принятого за начальный (рис. 3.1).

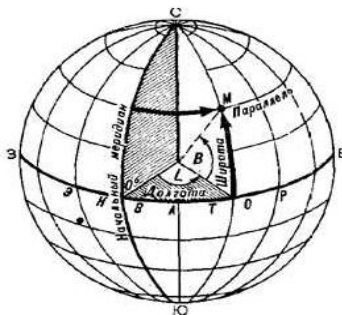


Рис.3.1

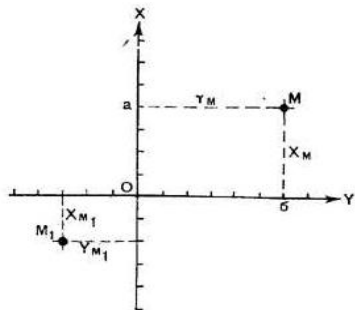


Рис. 3.2

Географическая широта – это угол, образованный плоскостью экватора и отвесной линией в данной точке земной поверхности. Величина угла показывает, насколько та или иная точка на земном

шаре севернее или южнее экватора. Если точка расположена в Северном полушарии, то ее широта называется северной, а если в Южном полушарии – южной.

На рисунке видно, что угол B соответствует широте точки M . Широта точек, расположенных на экваторе, равна 0° , а находящихся на полюсах (Северном и Южном) – 90° .

Географическая долгота – угол, образованный плоскостью начального меридиана и плоскостью меридиана, проходящего через данную точку. За начальный принят меридиан, проходящий через астрономическую обсерваторию в Гринвиче (близ Лондона). Все точки на земном шаре, расположенные к востоку от начального (Гринвичского) меридиана до меридиана 180° , имеют восточную, а к западу – западную долготу. Следовательно, угол L является восточной долготой точки M .

Известно, что сторонами рамок листов топографических карт являются меридианы и параллели. Географические координаты углов рамок подписываются на каждом листе карты.

Плоские прямоугольные координаты представляют собой линейные величины, определяющие положение точек на плоскости относительно установленного начала координат.

В общем случае за начало координат принимается точка пересечения двух взаимно перпендикулярных линий, называемых осями координат. Вертикальная ось называется осью x (X), а горизонтальная – осью y (Y). Положение точки определяется отрезками осей координат Oa и Ob или кратчайшими отрезками (перпендикулярами) от определяемой точки до соответствующих осей координат (X_m и Y_m). В нашем примере размер отрезка X_m равен 4 делениям, а отрезка Y_m – 6 делениям. Следовательно, прямоугольные координаты точки M будут: $X=4$, $Y=6$.

Значения величин X считаются положительными вверх (на север) от линии OY (оси Y) и отрицательными вниз от нее (рис. 3.2). Значения величин Y считаются положительными вправо (восточнее) от линии OX (оси X), отрицательными – влево от нее (западнее).

Применение системы плоских прямоугольных координат в топографии имеет некоторые особенности, вызванные шарообразной формой Земли, которая не может быть изображена на плоскости

без разрывов и искажений. Поэтому ее условно разделили на равные части, ограниченные меридианами с разностью долгот 6° , которые называли координатными зонами (рис. 3.3). Счет зон ведется от Гринвичского (начального) меридиана к востоку от 1 до 60.

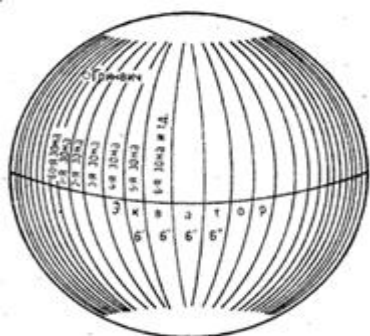
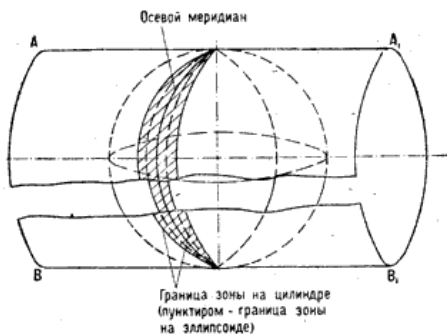


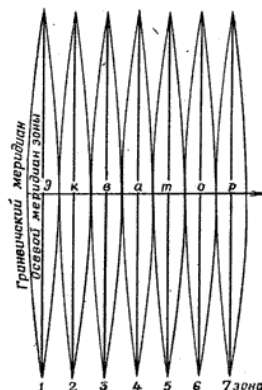
Рис. 3.3

Чтобы представить, как получается на плоскости изображение зон, вообразим цилиндр, который касается осевого меридиана одной из зон глобуса (рис. 3.4 а). Зону спроектируем по законам математики на боковую поверхность цилиндра так, чтобы при этом сохранилось свойство равноугольности изображения (равенство всех углов на поверхности цилиндра их величине на глобусе). Затем спроектируем на боковую поверхность цилиндра все остальные зоны, одну рядом с другой. Разрезав далее цилиндр по образующей AA₁ или BB₁ и развернув его боковую поверхность в плоскость, получим изображение земной поверхности на плоскости в виде отдельных зон (рис. 3.4 б).

Рис. 3.4



а



б

Рис. 3.4

В каждой зоне за вертикальную ось координат (ось X) принят осевой меридиан. Горизонтальной осью Y во всех зонах является

линия экватора. Пересечение осевого меридиана каждой зоны с экватором принято за начало координат.

Чтобы не иметь дело с отрицательными цифрами, условились считать координату 'у' в точке 0 (начало координат) равной не нулю, а 500 км. Общая протяженность зоны по экватору около 700 км. Поэтому при любом положении точки относительно среднего осевого меридиана зоны значение ее координаты 'у' будет положительным. Таким образом, точка 0 (начало координат) имеет координаты $x=0$, $y=500$ км. Имея в виду, что значение координаты 'у' осевого меридиана равно 500 км, следует запомнить, что все точки, координата 'у' которых более 500 км, расположены к востоку от осевого меридиана, а точки, имеющие координату 'у' меньше 500 км, — к западу от него.

Для того чтобы указать зону, в которой расположен объект, при определении его координат условились номер зоны писать при координате 'у' первыми цифрами, за которыми следует шестизначное число, показывающее значение координаты 'у' в метрах. Например, если точка А (рис. 3.5), расположенная в 12 зоне, находится к западу (влево) от осевого меридиана на удалении 191345 м, то ее координата 'у' имеет значение 12 308655, где число 12 обозначает номер зоны, а величина 308655 есть результат операции $(500000-191345)$. Если точка А находится на удалении от оси Оу в 6 081 км 462,5 м, то ее координата 'х' имеет значение 6 081 462,5.



Рис. 3.5

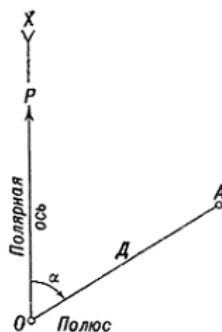


Рис. 3.6

Полярные координаты – компоненты системы координат, состоящей из точки O – начала координат, или полюса, и начального направления OP , называемого полярной осью (рис. 3.6). Положение точки A на местности или на карте в этой системе определяется двумя координатами: углом положения, который измеряется по ходу часовой стрелки от полярной оси до направления на определяемую точку A (от 0 до 360°), и расстоянием $OA=D$.

Занятие 8

ДВУМЕРНО-АССОЦИАТИВНЫЙ МЕХАНИЗМ ЗАЩИТЫ КАРТОГРАФИЧЕСКИХ ДАННЫХ

Для защиты данных ГИС может быть использован стохастический алгоритм шифрования, основанный на двумерно-ассоциативном механизме маскирования [3].

Лекция 8. Краткие сведения по алгоритму

В основе работы алгоритма лежит возможность ассоциации объектов по различию, т.е. достаточность сохранения некоторых признаков объектов для их последующей однозначной идентификации.

Рассматриваемый пример. Пусть множество из $\gamma = 10$ объектов представлено булевыми матрицами-эталонами размерностью $m \times (2m - 1)$, где $m = 3$ (рис. 3.7).

001	111	111	101	111	001	111	111	111	111
011	101	010	101	100	010	010	101	101	101
101	101	111	111	111	111	100	111	111	101
001	010	010	001	001	101	100	101	010	101
001	111	100	001	111	111	100	111	100	111

Рис. 3.7

На рис. 3.8 для наглядности идеальные эталоны показаны в форме почтовых индексов. Кругом отмечены дихотомизирующие биты идеальных эталонов, по которым возможна их однозначная идентификация. Информацию о локализации дихотомизирующих битов можно представить булевыми матрицами-масками (рис. 3.9).

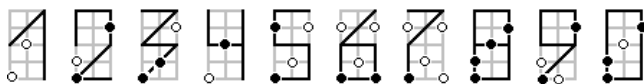


Рис. 3.8

000	000	000	000	000	000	000	000	000	000
000	001	001	000	101	101	101	001	001	001
010	000	000	010	000	000	000	010	000	010
000	100	010	000	010	010	010	100	100	100
100	110	100	100	100	101	101	100	110	100

Рис. 3.9

Число вариантов наборов масок (секретных ключей) для нашего случая составляет примерно $3 \cdot 10^{12}$. Это число растет с увеличением величин m и γ . На основе указанных идеальных эталонов и ключа можно получить троичные эталоны (рис. 3.10).

---	---	---	---	---	---	---	---	---	---
---	--1	--0	---	1-0	0-0	0-0	--1	--1	--1
-0-	---	---	-1-	---	---	---	-1-	---	-0-
---	0--	-1-	---	-0-	-0-	-0-	1--	0--	1--
0--	11-	1--	0--	1--	1-1	1-0	1--	10-	1--

Рис. 3.10

Порядок следования приведенных троичных эталонов соответствует указанному порядку объектов и масок. Запись “-” означает безразличное значение соответствующего элемента эталона. При шифровании места троичных эталонов с данной записью подвергаются стохастическому воздействию помех. Например, чтобы зашифровать первые три объекта из рис. 3.7 (код “123”) на рассмотренном ключе, надо взять соответствующие этому коду троичные эталоны и заменить все записи “-” в них на случайные бинарные значения. Вариант из возможных результатов шифрования выбранного кода показан на рис. 3.11.

Расшифрование ведется побитовым сравнением ключей и эталонов с зашифрованными объектами.

001	010	010
101	001	000
100	001	101
010	001	010
001	110	101

Рис. 3.11

Отметим, что алгоритм, на основе которого генерируется секретный ключ, получает маски, каждая из которых содержит от 2 до 6 единиц. Единицы в масках распределены по совокупному контуру

всех объектов (рис. 3.12 а, б – совокупные контуры на рассматриваемом множестве объектов для $m = 3$ и 7 соответственно).

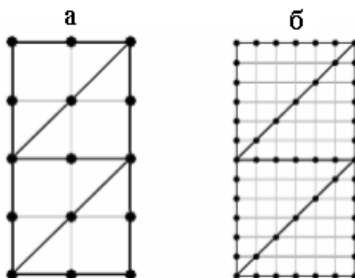


Рис. 3.12

Генерация набора масок осуществляется случайным образом для множества объектов из условия максимизация “числа степеней свободы”, т.е. числа записей “–” в троичных эталонах, которым можно приписать произвольные значения из $\{0, 1\}$. Знание масок является достаточным условием правильной идентификации объектов.

Стоимость шифра. Одной из важнейших характеристик качества шифра служит количество информации об исходном тексте, которое злоумышленник может извлечь из перехваченного шифротекста. Оно находится как разность между априорной и апостериорной неопределенностью исходного сообщения

Эта величина всегда неотрицательна. Показателем здесь является то, насколько уменьшится неопределенность исходного текста при получении соответствующего шифротекста по сравнению с априорной неопределенностью, и не станет ли она меньше минимально допустимой величины.

В наилучшем для разработчиков шифра случае обе эти неопределенности равны. Это означает, что злоумышленник не может извлечь никакой полезной для себя информации об открытом тексте из перехваченного шифротекста. Иными словами, знание шифротекста не позволяет уменьшить неопределенность соответствующего открытого текста, улучшить его оценку и увеличить вероятность его правильного определения. Шифры, удовлетворяющие данному условию, называются *абсолютно стойкими* или *совершенными шифрами*. Зашифрованные с их применением сообще-

ния не только не могут быть дешифрованы в принципе, но злоумышленник даже не сможет приблизиться к успешному определению исходного текста, т.е. увеличить вероятность его правильного дешифрования.

Говоря о безусловной стойкости, мы следуем критерию совершенной секретности К. Шеннона в следующей логической трактовке [4].

Если передача любого сообщения некоторого множества априорно равноправдоподобна и в итоге применения к каждой шифрограмме всевозможных ключей получаем равномошное множество апостериорно равноправдоподобных результатов распознавания, то использованный шифр обладает свойством совершенной секретности.

Исследованиями, проведенными в работе [5], подтверждена ГИПОТЕЗА. *Подходящая рандомизация (шифрограмма) для обеспечения безусловной стойкости в случае больших оснований ($\gamma = 10$) всегда существует.*

Это равносильно утверждению о принципиальной достижимости безусловной стойкости рассматриваемого шифра.

Поиск подходящей рандомизации для шифруемого кода на выбранном ключе заключается в следующем. Для некоторой рандомизации организуется перебор случайного подмножества ключей, размер которого задается пользователем (т.е. *полный перебор* ключей не проводится), и осуществляется идентификация по каждому набору.

Если на полученном множестве идентификаций для данной шифрограммы не будет выявлен хотя бы один из всевозможных кодов, то текущая рандомизация не удовлетворяет критерию совершенной секретности К. Шеннона. В таком случае осуществляется генерация следующей рандомизации.

Этот процесс продолжается до тех пор, пока не будет выявлена подходящая рандомизация, являющаяся основой получения шифрограммы. Временные затраты на получение шифрограммы уменьшаются с ростом m и увеличиваются с ростом γ .

При больших объемах данных использование рассматриваемого подхода к шифрованию связано с чрезмерными временными затратами. Поэтому в качестве сферы исследований была выбрана тематическая картография. Данные на таких картах не столь объемны, так что кодирование объектов картографии не вызывает "подавляющих" технических трудностей.

Практика 8. Лабораторная работа

1. Рассмотреть механизм шифрования с использованием программы “DemoCipher.exe” (папка “Демонстрация шифра”). Проанализировать зашифрованные объекты при изменении масок и помехи. Общий вид окна после запуска программы показан на рис. 3.13. При демонстрации параметры γ и m равны 10 и 8 соответственно.

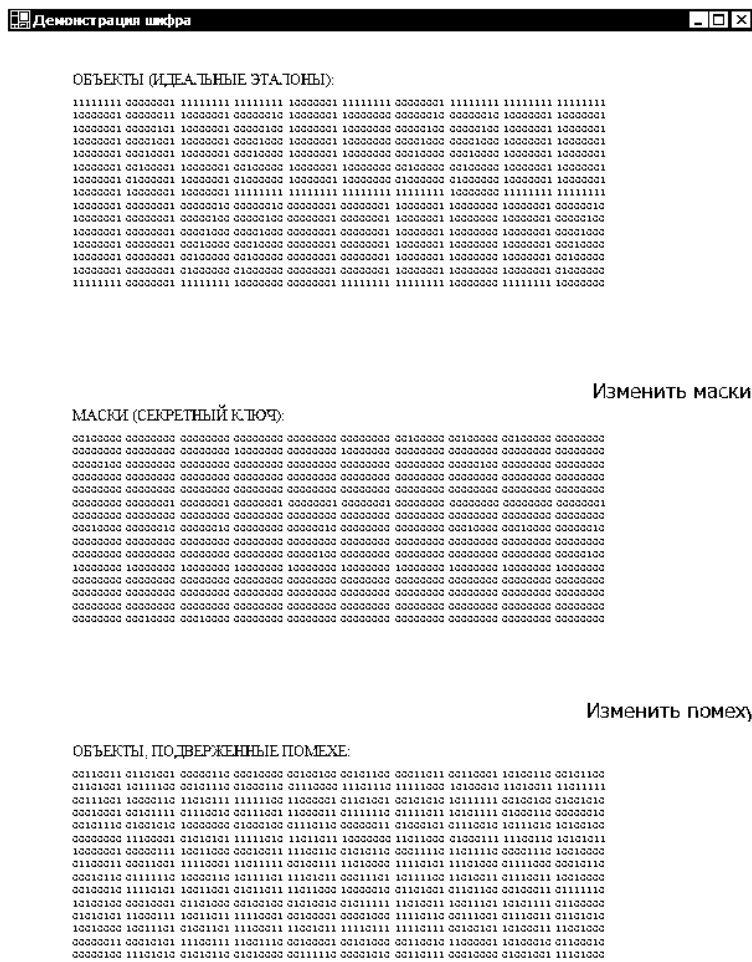


Рис. 3.13

2. Создать модуль расшифрования зашифрованных объектов при заданных идеальных эталонах (файл “Демонстрация шифра\data\objs.txt”) и секретном ключе (файл “Демонстрация шифра\data\msks.txt”).
3. Расшифровать зашифрованные объекты с использованием созданного модуля при ложных ключах.
4. Сделать выводы о проделанной работе.

Занятие 9

ПАРАЛЛЕЛЬНАЯ СУБД Security Map Cluster

Для формирования защищенной картографической базы данных (ЗКБД) исходные тематические слои карты, представленные в растровом формате (или в бумажном виде) и содержащие исключительно точечные объекты, подвергаются процедурам позиционирования и кластеризации (разбиению на кластеры-фрагменты). Эти процедуры связаны между собой. Их описание приводится ниже.

Лекция 9. Описание процедур и функционирование

Формирование ЗКБД. Пусть число градаций (значений кодов) объектов и их координат Γ равно γ^k , где k - количество разрядов в кодовом слове. Тогда например, если при $\gamma = 10$ число градаций координат равно 1000, то $k = 3$, т.е. любая координата будет представлена некоторым трехразрядным десятичным кодом.

На карту рис.3.14 нанесены локальная и глобальная координатные сетки. X и Y – максимальные значения координат x , y картографируемого массива, ε – заданная погрешность определения координат объектов. Соответственно шаг локальной координатной сетки равен 2ε . Значение Γ (вместе с γ и k) выбираем из условия равенства числа градаций в локальной и глобальной областях. Линейный размер фрагмента (длина стороны квадрата) $C = (2\varepsilon) \Gamma_{x,y}$. Полагая $X = Y = A$, получим необходимое условие представления координаты объекта суммой

КООРДИНАТА (x либо y) = ГЛОБ. КООРД. + ЛОК. КООРД.

в виде: $A/\Gamma_{x,y} \leq C$. Соответственно $\Gamma_{x,y} \geq \sqrt{A/(2\varepsilon)}$.

При формировании очередного фрагмента на карте случайным образом выбирается объект, не содержащийся ни в одном из ранее сформированных фрагментов. Глобальные координаты ле-

вого угла ячейки глобальной координатной сетки, в которой расположен выбранный объект, будут отвечать глобальным координатам создаваемого фрагмента. Вновь создаваемому фрагменту будут принадлежать все объекты, которые он покрывает, за исключением тех, которые содержатся в уже имеющихся фрагментах.

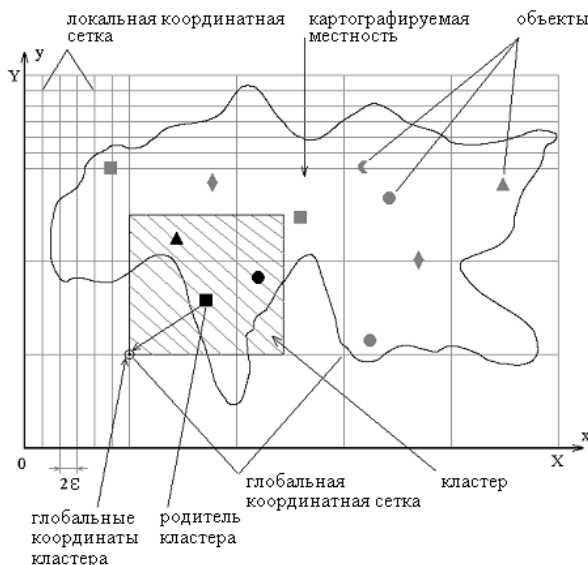


Рис. 3.14

Таким образом, объект, лежащий в области пересечения фрагментов, относится к фрагменту, который сформирован ранее остальных. Формирование фрагментов не завершится до тех пор, пока множество созданных фрагментов не покроют все объекты на карте. Каждому фрагменту присваивается свой номер. Нумерация фрагментов отвечает порядку их формирования.

После применения к исходной карте рассмотренных процедур, строится база данных формата СУБД MySQL для хранения данных ГИС. Выбор данного типа СУБД обусловлен рядом факторов: хорошая производительность, масштабируемость, надежность, простота использования и внедрения, низкие совокупные затраты, поддержка открытой и модульной разработки. Построенная база данных состоит из трех сущностей: Themes (Темы), Frames (Фрагменты), Objects (Объекты). Диаграмма “сущность-связь” этой базы данных, а также примеры таблиц приведены соответственно на рис. 3.15 и в табл. 3.1-3.3. Заметим, что подчеркнутые

тые атрибуты являются первичными ключами. Например, st является первичным ключом таблиц Frames и Objects.

Схема базы данных :

Themes [Темы слоев карт] (id_theme [Идентификатор темы]: integer, code_theme [Код темы]: text);

Frames [Фрагменты карт] (st [Счетчик-идентификатор]: integer, id_theme [Идентификатор темы]: integer, num_frame [Номер фрагмента]: integer, coord_x [Координата x фрагмента]: text, coord_y [Координата y фрагмента]: text);

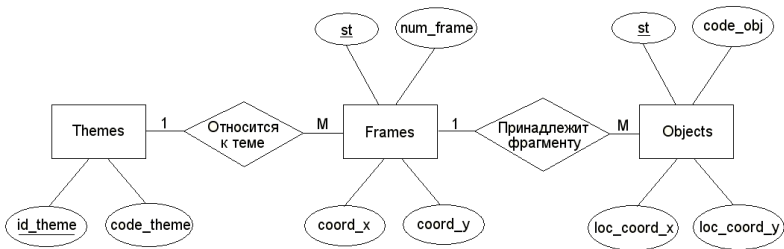


Рис. 3.15

Таблица 3.1

Themes	
id_theme	code_theme
1	001

Таблица .3.2

Frames				
st	id_theme	num_frame	coord_x	coord_y
1	1	1	005	001
2	1	2	004	003
3	1	3	005	002

Таблица .3.3

Objects					
st	id_theme	num_frame	code_obj	loc_coord_x	loc_coord_y
1	1	1	001	300	400
2	1	1	001	300	999
3	1	1	001	999	500
4	1	1	002	600	600
5	1	1	004	200	200

Objects [Объекты] (st [Счетчик-идентификатор]: integer, id_theme [Идентификатор темы]: integer, num_frame [Номер фрагмента]: integer, code_obj [Код объекта]: text, loc_coord_x [Координата x объекта во фрагменте]: text, loc_coord_y [Координата y объекта во фрагменте]: text).

При шифровании базы данных все значения в столбцах code_theme, coord_x, coord_y, code_obj, loc_coord_x, loc_coord_y заменяются соответствующими зашифрованными кодами.

Шифрование картографической базы данных ведется параллельно на узлах вычислительного кластера. Для распараллеливания программ по узлам кластера применяется библиотека передачи сообщений WMPI 1.3.

Алгоритм шифрования картографической базы данных. Пусть $q \geq n$ – общее число кодов, подлежащих шифрованию, n – число узлов кластера.

Алгоритм :

1. Главный процесс на управляющем узле кластера считывает из базы данных 'n' кодов и рассылает их процессам-шифровальщикам, запущенных на всех узлах. После рассылки кодов начинается их параллельное шифрование.
2. Процесс-шифровальщик, завершивший шифрование, передает главному процессу шифrogramму и код, которому она соответствует, и переходит в режим ожидания.
3. Главный процесс заменяет код в соответствующей ячейке базы данных, полученной шифrogramмой, уменьшает q на единицу и далее, если $q > 0$, передает ожидающему процессу очередной код. Иначе ожидающий процесс заканчивает работу. Шифрование базы данных завершится тогда, когда все процессы-шифровальщики закончат свои работы.

Вычислительными экспериментами установлено, что шифрация базы данных любой размерности на одной ПЭВМ происходит приблизительно в 'n' раз дольше, чем на вычислительном кластере, состоящем из 'n' таких ПЭВМ [5].

Перед шифрованием картографической базы данных фрагменты дополняются пустыми (реально не существующими) объектами, которые шифруются вместе с остальными. Код пустого объекта выбирается случайным образом из множества тех кодов объ-

ектов, которые не были задействованы на тематическом слое. Места локализации пустых объектов определяются случайно. После добавления пустых объектов должны соблюдаться следующие требования:

- каждый тематический слой карты включает все коды объектов;
- сумма пустых и непустых объектов в каждом фрагменте любого слоя карты равно числу N , где N не превышает максимально возможное число объектов во фрагменте;
- суммарное число вводимых пустых объектов по каждому слою карты минимально без нарушения начальных требований.

Кластеризация и размещение на каждом тематическом слое всевозможных кодов объектов является эффективным методом борьбы против криптографических атак, связанных с ассоциациями с картой местности.

Модуль управления. Необходимым условием эффективного управления защищенной картографической БД является равномерное распределение ЗКБД по узлам вычислительного кластера – платформы системы. Для выполнения этого условия создан программный модуль, описание работы которого приводится ниже.

Пусть число узлов кластера равно ' n ', и ЗКБД хранится на главном узле кластера (host-узле). Тогда модуль, запущенный на host-узле, распределяет кортежи отношения Objects, содержащего $M > n$ фрагментов карты, подчиненным узлам (slave-узлам) таким образом, что по окончании распределения каждый slave-узел содержит M/n различных фрагментов. Если M не кратно ' n ', то разница в количестве хранимых фрагментов между slave-узлами не превышает единицы. Slave-узлы хранят полученные кортежи в таблице Objects с той же структурой, что и в таблице Objects host-узла. Информация о том, на каких узлах расположены фрагменты, сохраняется в таблице Frames добавлением в нее столбца ip_address. Перед завершением своей работы модуль удаляет таблицу Objects на host-узле и рассылает таблицу Frames slave-узлам.

Реализованные процедуры. На вычислительном кластере реализованы процедуры с точечными объектами: добавление, удаление и визуализация прямоугольной области слоя карты.

- *Процедура добавления объекта.*

1. Проверить, не совпадают ли координаты добавляемого объекта q с координатами существующих объектов. Если совпадают, то процедуру добавления следует прервать.
2. Определить фрагмент Q , в который добавляется q . Если такого фрагмента не существует, то его требуется создать, заполнить непустым объектом q и необходимым числом пустых объектов, после чего завершить алгоритм.
3. Добавить q во фрагмент Q .
4. Проверить, имеется ли в Q хотя бы один пустой объект. Если имеется, то удалить в Q любой пустой объект. Иначе дополнить каждый фрагмент, кроме Q , одним пустым объектом. Данный шаг необходим для соблюдения равного числа объектов в каждом фрагменте.

- *Процедура удаления объекта.*

1. Определить фрагмент Q , в котором находится удаляемый объект q . Если такого фрагмента не существует, то процедуру удаления следует прервать.
2. Проверить, является ли q единственным непустым объектом в Q . Если является, то фрагмент Q удаляется. Иначе q заменяется пустым объектом.

Основным фактором, снижающим скорость исполнения описанных процедур, является длительное время шифрации кодов объектов и их координат. Поэтому шифрование данных в ходе выполнения процедур происходит на кластере.

Для визуализации участка тематического слоя проводится сбор картографической информации с узлов кластера.

- *Процедура сбора.*

1. Задаются координаты прямоугольной области карты.
2. На каждом узле кластера запускается MPI-процесс.
3. Главный процесс, запущенный на управляющем узле, создает коммуникационную группу, в которую входят процессы тех узлов, на которых хранятся нужные фрагменты.
4. Процессы коммуникационной группы параллельно выполняют SQL-запросы со встроенными функциями расшифрования, обращаясь к локальным частям ЗКБД. Процессы, не вошедшие в группу, завершаются.

5. Главный процесс собирает результаты обработки запросов со всех процессов созданной группы и передает данные на визуализацию.

Для оценки времени выполнения вышеописанных процедур создана тестовая ЗКБД, которая содержит один тематический слой с числом фрагментов 71, $N = 28$. На тестовом слое задействовано 16 реальных объектов. Эксперименты проводились при $\Gamma = 1000$, $\gamma = 10$, $m = 18$.

Работа процедур с тестовой ЗКБД проводилась на вычислительном кластере при различном числе узлов, объединенных сетью Gigabit Ethernet посредством коммутатора D-LINK DGS-1016D. Каждый из узлов имеет двухъядерный процессор Intel(R) Core(TM)2 CPU частотой 1,87 ГГц и оперативную память 3 Гб.

Результаты тестирования процедур представлены в табл. 3.4. Отметим, что по каждой процедуре проведено 50 испытаний. При тестировании процедуры добавления рассмотрены 3 случая:

1. Добавление объекта сопровождается удалением пустого объекта;
2. Добавление объекта требует создание нового фрагмента;
3. Добавление объекта требует увеличение числа N на 1.

Таблица 3.4

Тип процедуры		Среднеарифметическое время выполнения процедуры, сек.			
		На одной машине	На различных конфигурациях кластера		
			4-х узловая	8-ми узловая	12-ти узловая
Добавление объекта	Случай 1	53,2	36,5	41,4	34,6
	Случай 2	1414,5	374,6	211	158,7
	Случай 3	3230,2	858,6	476,9	358,6
Удаление объекта		56,5	32,1	37,6	34,3
Сбор данных для визуализации всей карты		3,6	2,4	1,7	1,4

Из таблицы видно, что выполнение процедур на вычислительном кластере происходит сравнительно быстрее, чем на одной машине. Так как при выполнении процедур удаления и добавления для 1-го случая число шифруемых кодов невелико, то масштабирование кластера не увеличивает скорость их работы. При выполнении всех оставшихся процедур кластер демонстрирует хорошую масштабируемость.

СУБД Security Map Cluster реализует многопользовательский режим работы и строится с применением идеологии “клиент-сервер”. Визуализация картографических данных на клиентской машине достигается применением ГИС MapInfo Professional. Подсистема визуализации снабжена пользовательским интерфейсом, посредством которого пользователь не только отображает картографические объекты, но и управляет системой. Сервером в данном случае выступает вычислительный кластер с архитектурой “без разделения ресурсов” (shared nothing). Упрощенная архитектура системы представлена на рис. 3.16.

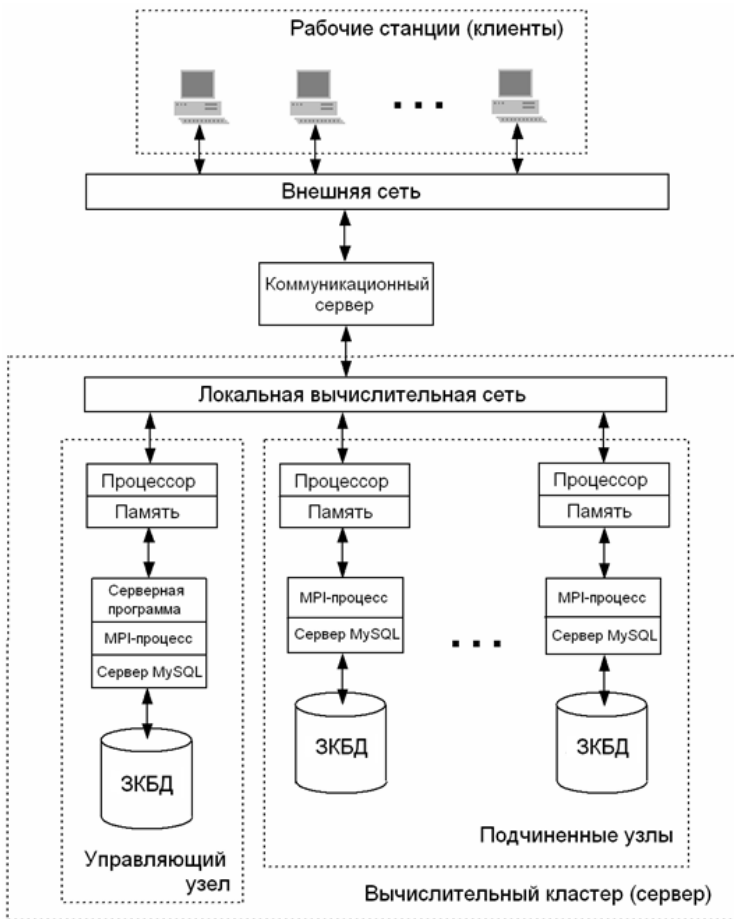


Рис. 3.16

Цикл работы клиента с сервером состоит из следующих этапов:

1. Пользователь запускает клиентскую программу, в которой указывает процедуру, которую необходимо выполнить, и задает входные параметры этой процедуры.
2. Клиентская программа запускает серверную программу на управляющем узле кластера и передает ей запрос пользователя.
3. Серверная программа выполняет требуемую процедуру и передает результаты работы клиентской программе.

В ходе своей работы серверная программа запускает MPI-процессы на узлах вычислительного кластера для решения подзадач шифрования и обработки SQL-запросов к частям ЗКБД.

Защита внешней сети не требуется, так как передача картографических данных клиенту происходит в зашифрованном виде.

Подсистема визуализации. Рассмотрим меню и диалоги, которые были встроены в ГИС MapInfo Professional для работы с защищенными картографическими базами данных.

ГИС MapInfo Professional является мощной средой для визуализации картографических данных [6]. Она совмещает преимущества обработки информации, которыми обладают базы данных (включая мощный язык запросов SQL), и наглядность карт, схем и графиков. MapInfo – открытая система. Благодаря языку программирования MapBasic возможно создание на базе MapInfo собственных ГИС. MapInfo обладает широкими функциональными возможностями, требующие достаточно объемного изложения. Поэтому подробное описание данной ГИС здесь не приводится.

Для начала работы с подсистемой визуализации после запуска MapInfo необходимо в меню “Программы” выбрать пункт “Запустить программу MapBasic...” и в появившемся диалоговом окне выбрать файл wbase.mbx. Экранная форма после запуска программы представлена на рис. 3.17.

В результате запуска программы к стандартному меню MapInfo добавляется пункт “Работа с ЗКБД”, содержащий пункты: “Представление точечных объектов”, “Добавление объекта”, “Удаление объекта”, “Конец работы”. При выборе пунктов “Представление точечных объектов”, “Добавление объекта”, “Удаление объекта” появляются диалоговые окна (рис. 3.18-3.20), в которые требуется

ввести запрашиваемые данные и нажать на кнопку “ОК” для выполнения процедуры. Для завершения работы с ЗКБД необходимо выбрать пункт “Конец работы”.

Отметим, что для корректного выполнения процедур в файл “private_key.txt”, хранящийся в исполняемых модулях программы, необходимо записать правильный секретный ключ. Однако при неверном ключе подсистема не теряет своей функциональности, что не позволяет криптоаналитику отсеивать ложные ключи по тому признаку, что подсистема завершает работу при неверном ключе.



Рис. 3.17

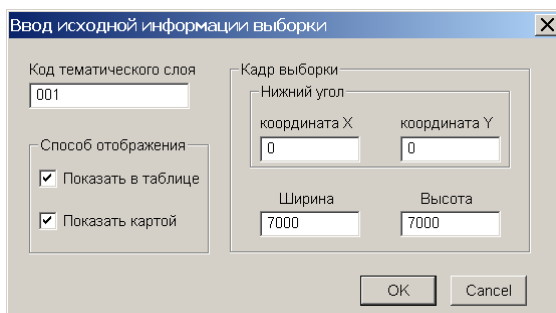


Рис. 3.18

Параметры добавляемого объекта

Код тематического слоя: 001

Код объекта: 000

Расположение

Координата X: 0

Координата Y: 0

OK Cancel

Рис. 3.19

Параметры удаляемого объекта

Код тематического слоя: 001

Код объекта: 000

Расположение

Координата X: 0

Координата Y: 0

OK Cancel

Рис. 3.20

Практика 9. Лабораторная работа

1. Проанализировать, как сформирована тестовая картографическая база данных (папка “Картографическая база данных”) на основе растровой карты (папка “Растровая карта”).

2. Проанализировать картографическую базу данных после добавления в нее пустых объектов. Для добавления пустых объектов использовать модуль “AddEmptyObjs.exe” (папка “Модуль, добавляющий пустые объекты”).

3. Зашифровать базу данных с применением модуля “Cipherng DB.exe” (папка “Модуль, шифрующий БД”). Проанализировать время шифрования картографической базы данных с увеличением числа узлов вычислительного кластера.

4. Распределить ЗКБД по узлам вычислительного кластера с использованием модуля “Distribution.exe” (папка “Модуль, распределяющий БД по узлам кластера”).

5. Выполнить процедуры визуализации карты, добавления и удаления объекта, используя СУБД Security Map Cluster.

6. Выполнить пункты 4, 5 при различном числе узлов вычислительного кластера и получить соответствующие временные оценки.

7. Сделать выводы о проделанной работе.

ЛИТЕРАТУРА К РАЗДЕЛУ III

1. *Каев А.* Системы координат в картографии и геодезии – Интернет-адрес: <http://www.firststeps.ru/gis/geodez/r.php?3>
2. *Бугаевский Л.М., Цветков В.Я.* Геоинформационные системы: Учебное пособие для вузов М.:2000.
3. *Райхлин В.А., Вершинин И.С., Глебов Е.Е.* К решению задачи маскирования стилизованных двоичных изображений //Вестник КГТУ им. А.Н. Туполева. 2001. №1. С. 42-47.
4. *Райхлин В.А.* Конструктивное моделирование систем – Казань: Изд-во ФЭН (Наука), 2005.
5. *И.С. Вершинин, Р.Ф. Гибадуллин, П.Е. Земцов.* Параллельные алгоритмы защиты бинарных объектов картографии //Моделирование процессов /Под. ред. В.А. Райхлина. *Труды Казанского научного семинара «Методы моделирования»*. Вып. 3. – Казань: Изд-во КГТУ, 2007. С.96-108.
6. *MapInfo Professional.* Руководство пользователя. – MapInfo Corporation Troy, New York. Перевод: Журавлев В.И, Колотов А.Ю., Николаев В.А., 2000г.