НИУ МЭИ

Кафедра ВМСиС

Вычислительные системы

Конспект лекций

Лектор к.т.н. проф. И. И. Ладыгин

СОДЕРЖАНИЕ

Лекция 1 4
1. Основные понятия в области вычислительных систем 4
1.1 Введение. Основные определения 4
Лекция 2 10
1.2 Понятие Фон-неймановской архитектуры 10
1.3 Парадигмы Фон-неймановской архитектуры. 12
1.4 Особенности Фон-неймановской архитектуры 12
Лекция 3 16
2. Эволюция способов обработки данных 16
Лекция 4 24
3. Уровни параллелизма при выполнении прикладных задач 24
4. Распараллеливание последовательных частей программ 28
Лекция 5 33
5. Оценка производительности ВС 33
Лекция 6 37
6. Методы оценки производительности ВС 37
7. Система тестов SPEC. Единицы измерения производительности ВС 39
Лекция 7 45
8. Классификация вычислительных систем 45
9. Режимы обработки данных ВС 57
9.1 Пакетный режим функционирования ВС 57
9.2 Режим разделения времени (потоковый режим) 58
9.3 ВС реального времени 61
Лекция 8 63
10. Системы класса MIMD (МКМД) 63
11. ВС с общей памятью (UMA) 65
12. Проблема когерентности в ВС с общей памятью 66
Лекция 9 68
13. ВС с распределенной памятью 68
Лекция10 70
14. Постановка задачи назначения 70
14.1 Модели представления задач 71
14.2 Алгоритм поиска критического пути графа 73
14.3 Применение стратегий назначения при решении задачи назначения 76
Лекция 11 77
14.4 Обеспечение надежности вычислений 77
14.5 Многозадачный режим функционирования Многопроцессорной Вычислительной Системы 79
Лекция 12 82
15. Системы коммутации 82
15.1 Шинные структуры 83
15.2 Матричные структуры 89
15.3 Кубические структуры 92
Лекция 13 96
15.4 Многоступенчатые коммутаторы 96
15.5 Сеть Омега - Коммутатор Omega ILLIAC-IV 97
15.6 Коммутационная сеть flip (сеть перестановок) 98
Лекция 14 100
16. ВС на основе векторных процессоров 100
Лекция 15 105

17. ВС класса ОКМД на основе матричных процессоров 105
17.1 Структурная схема ILLIAC- IV 107
17.2 Нисходящее проектирование матричных процессоров 110
17.3 Оценка эффективности ВС на основе матричных процессоров 115
Лекция 16 118
18. Супер-ЭВМ семейства CRAY (ВС с ОП) 118
18.1 CRAY C-90 122
18.2 СуперЭВМ NEC SX (1÷6) 124
Лекция 17 128
19. ВС на основе ассоциативных процессоров 128
19.1 Введение 128
19.2 Ассоциативные процессоры 131
19.3 Ассоциативные процессоры с пословной организацией 132
19.3.1 Базовая структура 132
19.3.2 Базовые операции 137
19.4 Ассоциативный процессор Staran 144
Лекция 18 148
20. Особенности программного обеспечения параллельных систем 148
20.1 Языки параллельного программирования 150
20.2 Особенности трансляторов параллельных систем 154
20.3 Операционные системы 155
Лекция 19 157
21. Процессоры высокопроизводительных вычислительных систем 157
22. Предшественники многоядерных процессоров 159
22.1 Особенности построения Elbrus 3M 161
22.2 Технология Hyper Threading 164
22.3 Оценка эффективности двухядерного процессора. 171
Лекция 20 174
23. Основы обработки графической информации 174
23.1 Процессор i860. Структурная схема. Регистры 176
Лекция 21 181
24. Вычислительная система «ЭЛЬБРУС» 181
24.1 Мультипроцессорный вычислительный комплекс «ЭЛЬБРУС-2» (МКМД с ОП) 184
24.1.1 Центральный процессор 185
24.1.2 Формат данных ЦП 188
24.1.3 Организация оперативной памяти 190
24.2 МВС «Эльбрус-3» (МКМД с ОП) 191
24.2.1 Основные принципы обработки данных в «Эльбрус-3» 193
24.2.2 Структурная схема ЦП «Эльбрус-3» 194
24.2.3 Структура распакованной команды «Эльбрус-3» 198

Лекция 1

1. Основные понятия в области вычислительных систем

1.1 Введение. Основные определения

Термин *«Вычислительная система» (ВС)* появился в 60-е годы, когда понятие «Электронная вычислительная машина» (ЭВМ) уже не отражало с одной стороны всего многообразия средств вычислительной техники, а с другой стороны не различало их по степени сложности. Поэтому интуитивно можно считать, что *вычислительная система* - это объект, относящийся к средствам вычислительной техники (СВТ) и имеющий более сложную организацию, чем ЭВМ. К сожалению, до настоящего времени сколько-нибудь удовлетворяющего всех разработчиков СВТ определения ВС не установлено. Однако существует необходимость хотя бы с методической точки зрения различать объекты, относящиеся к понятию ЭВМ и к понятию ВС, при этом понимая, что чёткой грани, разделяющей эти объекты провести невозможно.

Попытаемся внести смысловой оттенок в определение ВС через разъяснение нашего понимания таких терминов, как «система», «структура», «архитектура», толкование которых само по себе вызывает много споров у специалистов по вычислительной технике. В данной области техники термин *«система»* используется чрезвычайно широко и имеет множество смысловых оттенков. Мы будем понимать под системой объект, представляющий собой единое целое, предназначенный для выполнения определенных функций и состоящий из множества связанных между собой элементов. Под элементами могут пониматься как аппаратные, так и программные средства. Следовательно, такие объекты, как ЭВМ и ВС, являются системами, предназначенными для автоматизированной обработки данных и представляющие собой совокупность программных и аппаратных средств. Для их обозначения используется понятие *«система обработки данных» (СОД)*. На рис. 1.1 представлены виды систем обработки данных, в том числе ВМ (синоним ЭВМ) – вычислительная машина, ВК – вычислительный комплекс.

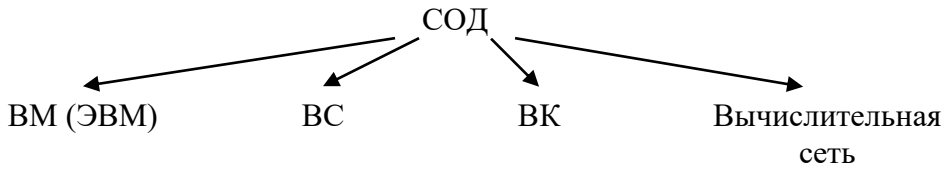


Рис. 1.1 Виды систем обработки данных

Структура СОД обычно определяется в виде совокупности трёх категорий: *множество аппаратных элементов*, входящих в состав системы, *множество связей*, объединяющих эти элементы, и *алгоритм функционирования системы*. В контексте проектирования аппаратных

средств вычислительной системы термин «структура» используется для описания принципа действия, конфигурации и взаимного соединения устройств системы.

Архитектура СОД может быть определена как совокупность тоже трёх категорий: *сущность информационных потоков*, например, существование двух потоков – команд и данных, *характер взаимодействия информационных потоков*, например, жесткая последовательность команд управляет обработкой данных, *способ обработки данных*, например, последовательный или параллельный. Следовательно, понятие архитектуры отображает обобщенное определение системы с точки зрения существующих в ней информационных потоков и способа их обработки.

Общность архитектуры разных ЭВМ и ВС обеспечивает их совместимость с точки зрения пользователя. Реализация конкретной архитектуры ЭВМ и ВС может быть различной: использование разных по физической природе элементов, применение оригинальных решений при разработке устройств системы, её структуры, что приводит, например, к различию систем по стоимости, производительности, но не меняет принципы обработки данных. В настоящее время можно выделить три вида понятий, связанных с термином «архитектура»: фоннеймановская архитектура, усовершенствованная фоннеймановская архитектура, нетрадиционная архитектура.

Первый вид определяет класс объектов с архитектурой, которая основана на использовании двух информационных потоков - команд и данных, с чисто последовательным характером обработки данных под управлением заранее составленной последовательности команд (программы). Понятие «усовершенствованная фоннеймановская архитектура» связано с реализацией различных методов распараллеливания потока обработки данных с целью увеличения производительности системы, но не нарушая принципа программного управления. В этом случае можно говорить о том, что этот вид архитектуры отличается от первого только увеличением числа потоков команд и данных в любых сочетаниях. Естественно, в зависимости от числа потоков команд и данных, примененного метода распараллеливания и характера управления множествами потоков «усовершенствование» архитектуры может идти различными путями, приводя к появлению новых типов архитектур в рамках данного вида.

Понятие «нетрадиционная архитектура» связано, в частности, с так называемыми потоковыми машинами, которые отличаются новым принципом управления вычислительным процессом и разделяются на два типа архитектур: управляемые потоком данных (нет счетчика команд; устройство управления сложное, обычно выполненное на АЗУ; программы нет как таковой) и управляемые запросами. При этом, по аналогии с фоннеймановской архитектурой, организация вычислений может быть «последовательной» (здесь этот термин используется только с точки зрения аналогии, так как по своей сути потоковые машины строятся с

использованием всех видов параллелизма, поэтому последовательные потоковые машины можно рассматривать как вырожденный случай) и параллельной.

Таким образом, определяя понятие «вычислительная система» с точки зрения архитектуры, можно сделать вывод о том, что этот класс СОД должен обладать либо усовершенствованной фоннеймановской архитектурой, либо нетрадиционной с параллельной организацией вычислений. При этом ЭВМ обладают либо фоннеймановской архитектурой, либо нетрадиционной с последовательной организацией вычислительного процесса.

Для того, чтобы дать более точное определение ЭВМ и ВС, необходимо определить их место среди таких объектов СОД, как ВК и вычислительная сеть. Для этого дадим следующие определения, введённые профессором кафедры ВМСиС Дерюгиным А.А [1] при участии автора.

1.*Процессор, центральный процессор, центральное процессорное устройство (обозначение на схемах P,CP,CPU)* – основная часть ЭВМ, предназначенная для обработки данных и управления этой обработкой в соответствии с последовательностью команд программы. Такой процессор по умолчанию является скалярным. В состав процессора входят:

а) арифметико-логическое устройство (ALU), состоящее из одного или нескольких блоков обработки числовых данных с фиксированной и плавающей запятой, блока логических операций, сдвига данных и др.;

б) устройства управления (CU – control unit), содержащее счетчик команд (IP – instruction pointer), регистр команд (инструкций,RGI), часто буфер команд, блок преобразования кода операций в коды микроопераций, т.е. в последовательности управляющих сигналов, соответствующих микрооперациям, как для управления работой процессора, так и для управления другими устройствами ЭВМ;

в) блок регистров, включая адресуемые регистры общего назначения (регистровую память RGM), регистры управления, в том числе регистр флагов (признаков) и др., а также неадресуемые регистры (дескрипторы и др.);

г) интерфейсный блок (IU), предназначенный для реализации связей процессора с другими устройствами ЭВМ. В состав этого блока входит, в частности, узел управления памятью (менеджер памяти), предназначенный для формирования из кода адреса операнда, указанного в команде, физического адреса ячейки памяти, в которой хранится требуемый операнд.

Виды процессоров

1.1. Суперскалярный процессор (SSP) – процессор, допускающий параллельное (одновременное) выполнение нескольких команд программы. Такой процессор может содержать несколько конвейеров обработки данных.

1.2. Векторный процессор (VP) – процессор, содержащий специальные блоки обработки данных, предназначенные для выполнения, как

скалярных операций, так и операций над векторами в соответствии с векторными командами, входящими в состав системы команд процессора, а также блок векторных регистров (VRG).

1.3. Матричный процессор (MP) – многопроцессорное устройство, предназначенное для выполнения операций над векторами и матрицами.

1.4. Ассоциативный процессор (AP) – специализированный процессор, реализованный на базе ассоциативного запоминающего устройства и предназначенный для одновременного выполнения операций над массивами данных последовательно по разрядам этих данных.

2. Кэш-память (CM) – быстродействующая буферная память небольшой ёмкости, в которой хранятся команды и данные, необходимые для использования в течение текущего интервала времени. Содержимое CM дублирует содержимое основной (большой) памяти (BM – base memory) и меняется в процессе работы ЭВМ автоматически аппаратным способом. Обмен данными между CM и BM производится строками (блоками). Обычно это 16…64 байта. В состав CM входят: собственно кэш-память и местное устройство управления, включая таблицу, в котрой хранятся адреса тех блоков, которые находятся в кэш-памяти в данный момент. В зависимости от «удаления» от процессора различают CM первого и второго уровней (L1и L2, L –level). Кэш-память первогог уровня может быть общей для команд и данных или раздельной для команд и данных. Кэш-память второго уровня, как правило, общая для команд и данных.

3. Процессорный модуль (PM), вычислительный модуль – устройство, содержащее процессор, кэш-память или, например, локальную память (LM).

4. Внутренняя память (IM – internal memory) – память, предназначенная для приёма, хранения и выдачи информации, непосредственно используемой (адресуемой) процессором. IM _ это RGM+CM+BM.

5. Внешняя память (EM – external memory) – память, предназначенная для длительного хранения больших объёмов информации. Для использования этой информации её необходимо переместить во внутреннюю память средствами операционной системы. Внутренняя память является буферной памятью по отношению к внешней памяти.

6. Многопроцессорный модуль (MPM) – функциональное устройство, содержащее вычислительные модули, соединённые с кэш-памятью второго уровня. Из PM и MPM могут быть построены многопроцессорные вычислительные системы (MBC) с двухуровневой или трёхуровневой общей внутренней разделяемой памятью.

7. Общая или разделяемая память МВС – это внутренняя память, к ячейкам которой может обращаться любой процессор, входящий в состав МВС.

8. Локальная память (LM) процессора, входящего в состав МВС, - это внутренняя память, к ячейкам которой может обращаться только данный процессор. Адресные пространства для общей и локальной памятей не пересекаются.

9. Коммутатор (SW) МВС – аппаратные средства, обеспечивающие взаимодействие между компонентами МВС. В состав коммутатора в явном или неявном виде входит арбитр.

В состав СОД могут входить так называемые *вспомогательные процессоры*. К ним относятся управляющие процессоры, обеспечивающие реализацию функций, служебных по отношению к самой СОД, сервисные процессоры для выполнения функций контроля и диагностики СОД, процессоры ввода-вывода, процессоры телеобработки данных и т.д.

Совокупность любых процессоров любого количества, оперативной памяти и каналов ввода-вывода назовём *центральной частью (ЦЧ)* СОД.

СОД, включающую в себя одну центральную часть, в состав которой входит один процессор обработки данных любого вида и возможно один или несколько вспомогательных процессоров р, обеспечивающих максимально эффективную работу первого, а также периферийные устройства, назовём *электронной вычислительной машиной (ЭВМ)*.

СОД, включающую в себя одну ЦЧ, но имеющую более одного процессора обработки данных любого вида, периферийные устройства, назовём *вычислительной системой (ВС)*.

СОД, включающую в себя несколько ЦЧ, объединенных по каналам ввода-вывода, и общие для них периферийные устройства, назовём *вычислительным комплексом (ВК)*.

Множество ЭВМ, ВС и ВК объединённых линиями связи, назовём *вычислительной сетью*.

Настоящее учебное пособие посвящено, в основном, отдельным вопросам, связанным с вычислительными системами с усовершенствованной фоннеймановской архитектурой. При анализе таких систем для количественной оценки их эффективности обычно используется такой показатель, как *производительность*. Приведем наше толкование данного показателя. При этом будем различать два взаимосвязанных вопроса: какое качество системы отображает это понятие и в чём можно измерить это качество. Отвечая на эти вопросы, можно определить производительность как характеристику вычислительной мощности системы, определяющую количество выполненной работы за единицу времени. Таким образом, если имеем в виду вычислительную систему, то под работой можно понимать решение задач. В этом случае производительность есть число задач, выполненных системой в единицу времени. Следовательно,

производительность системы будет различной для задач различных классов. Существует большое число методов оценки производительности ВС, связанных с тем или иным классом задач, однако главные из них основаны на использовании специально созданных тестов. Для того, чтобы оценить производительность системы, не привязываясь к классу задач, иногда используют такой показатель, как *быстродействие*. Хотя этот показатель отражает только техническую сторону вычислительной системы, а иногда характеристику только основного элемента системы - процессора, он достаточно широко распространен. Если в первых ЭВМ быстродействие измерялось в количестве коротких операций в секунду, например, типа «регистр-регистр» или «сложение с фиксированной точкой», то современные ЭВМ и ВС характеризуются числом выполняемых операций с плавающей точкой в секунду, которые получили название флопов (FLOPS). Для специализированных ВС используются другие единицы измерения, например характерные для них операции, транзакции и т.д.

Данный курс лекций поможет студентам ответить на такие вопросы как: зачем нужны ВС всё большей и большей производительности, как видоизменяются структура и архитектура процессора и самой ВС с целью повышения эффективности использования их аппаратных средств, какими свойствами должно обладать их программное обеспечение, какие новые идеи существуют в области разработки высокопроизводительных вычислительных систем. И как показывает опыт развития средств вычислительной техники, вопросов, на которые пока нет ответов, появляется всё больше и больше.

Лекция 2

1.2 Понятие Фон-неймановской архитектуры

Джон фон Нейман (Янош 1903-1957) родился в Будапеште, а с 1930 г. жил и работал в США. По профессии математик и физик. Предложил принцип обработки данных и архитектуру программного автомата, который получил название неймановской машины. Построение ЭВМ, основанной на этом принципе, базируется на идее автоматизации вычислений под управлением программы (последовательность инструкций) и использует две фундаментальные мысли.

Первая из них состоит в том, что программа вводится через те же внешние устройства и хранится в той же памяти, что и исходные данные. Это обеспечивает оперативную перестройку машины с одной задачи на другую без внесения каких0либо изменений в схему машины и её коммутацию, делая машину универсальным вычислительным инструментом. Термин универсальная ЭВМ широко использовался в последующие годы.

Вторая состоит в том, что инструкции, составляющие программу вычислений, закодированы в виде чисел и по форме ничем не отличаются от тех чисел, с которыми оперирует машина. Это дает возможность при выполнении некоторой инструкции прочесть другую или, в частном случае, ту же инструкцию как число, переслать её в арифметическое устройство, произвести там с ней некоторые операции и вернуть на своё место в запоминающем устройстве (памяти) в преобразованном виде. Когда в следующий раз устройство управления обратится к данной ячейки памяти за инструкцией, а не за операндом, то исполняться будет уже не исходная, а преобразованная инструкция. Таким образом, при исполнении некоторой программы может одновременно происходить её преобразование, либо формирование новой программы. Следовательно, можно сказать, что неймановская машина по своей сути обладает элементами интеллектуальной системы.

Прежде чем рассматривать последующие усовершенствования неймановской машины, напомним принципы её действия. Основными элементами базового варианта неймановской машины, представленного на рис. 1.2, являются:

- программный счётчик (ПС);
- регистр команд (РК);
- схема управления (СУ);
- арифметико-логическое устройство (АЛУ);
- аккумулятор (А);
- схемы представления чисел в дополнительном коде с регистром сдвига (СПЧ);
- память инструкций (команд) и данных (П);
- регистр адреса (РА);
- регистр данных (РД);

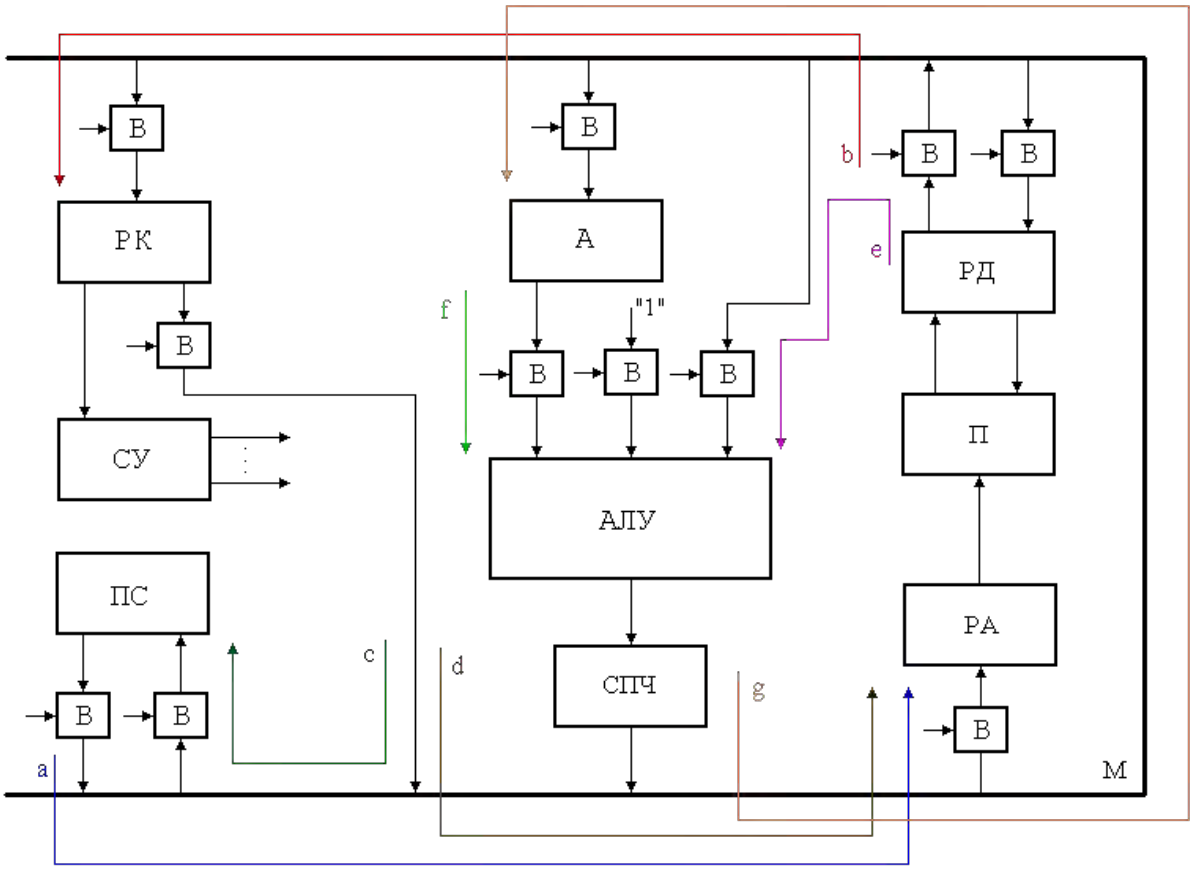


Рис. 1.2 Базовый вариант неймановской машины

- совокупность вентильных групп (В), на которые поступают управляющие сигналы из СУ;
- внутренняя машинная магистраль (М).

Процесс обработки данных в такой машине чисто последовательный и выполнение каждой команды разбивается на строго определенный ряд следующих друг за другом действий. Если за точку отсчёта принять такое состояние машины, при котором в ПС установлен адрес следующей команды, то эти действия (шаги) следующие:

a). по адресу команды, находящемуся в ПС, происходит обращение к П с использованием РА, после чего сама команда оказывается в РД;

b). код команды заносится из РД в РК;

c). в случае команды условного (безусловного) перехода, формирование номера следующей команды и занесение его в ПС;

d). в случае исполнительной команды, обращение в память за операндом по адресу, расположенному в коде команды или по адресу, вычисленному специальной схемой;

e). занесение операнда из РД в АЛУ;

f). занесение операнда из А в АЛУ;

g). занесение результата операции из СПЧ в А.

1.3 Парадигмы Фон-неймановской архитектуры.

Существуют две парадигмы, на которых базируется так называемая неймановская архитектура.

1) Входные данные и инструкции поступают по одному и тому же каналу и хранятся в одной и той же памяти.

2)Управление выполнением последовательности команд осуществляется программным счётчиком (ПС).

Сама команда, как это было отмечено выше, выполняется то же в виде последовательности шагов, число которых в современных ЭВМ может отличаться от приведённого примера, но суть их последовательности при выполнении одной команды остаётся неизменной.

Следовательно, главное определение неймановской машины – это последовательная машина.

1.4 Особенности Фон-неймановской архитектуры

Анализ особенностей функционирования неймановской архитектуры показывает, что чисто последовательный характер обработки данных приводит к тому, что в каждый момент времени (применительно к процессору момент времени должен рассматриваться как машинный такт) функционирует только определённая часть аппаратуры, выполняющая соответствующие действия в рамках одного из шагов выполнения команды. Остальная часть аппаратуры простаивает. Именно этот вывод и до сих пор служит для широкого поля деятельности в совершенствовании технических характеристик систем обработки данных. С другой стороны, анализируя прикладные программы, созданные для реализации на ЭВМ, можно согласиться с выводами ученых, например, Прангишвилли И. В., которые утверждают, что любая программа содержит такую часть, которую можно распараллелить. Именно в этом состоит основное зерно, лежащее в основе усовершенствования неймановской архитектуры.

Рассматривая в целом такую СОД как ЭВМ с неймановской архитектурой, можно привести такие основные её проблемы. Здесь под проблемами понимаются её узкие места, не позволяющие эффективно использовать аппаратно-программные средства ЭВМ.

1. Программные коды (команды) вместе с данными хранятся в одной и той же памяти.

   Это приводит к тому, что в процессе выполнения одной команды необходимо несколько раз обратится последовательно к одной памяти, что значительно замедляет процесс обработки данных.
2. Линейное пространство адресов, которым присваиваются порядковые номера 0, 1, 2, 3... .

Это свойство ограничивает эффективность применения памяти с модульной организацией.

3. Каждая программа выполняется последовательно, начиная с самой первой команды, если нет специального указания (команды перехода). Это свойство ограничивает возможности использования характера параллельности выполняемых программ.
4. Последовательное выполнение обработки в неймановской архитектуре приводит к последовательному изменению состояния машины путём изменения содержимого памяти. Это явление носит название *побочного эффекта*. Поскольку вычисления выполняются с побочным эффектом, то результат операции нужно каждый раз записывать в память.
5. Отсутствие различий в машинном представлении данных и команд. В этом случае, в частности, возникают проблемы, усложняющие разработку программного обеспечения. Наличие разрыва между понятиями операций и их объектов на языке программирования высокого уровня и понятиями операций и их объектов, определяемыми архитектурой компьютера, называется *семантическим разрывом.*
6. Отсутствие различий в семантике данных. Иначе говоря, если на языке высокого уровня можно описать различные типы данных, то в неймановской архитектуре разницы в представлении типов данных нет. Поэтому каждый раз, когда необходимо выполнить те или иные операции над данными нужно либо программно, либо аппаратно проконтролировать их тип.

Какие существуют способы для "смягчения" влияния узких мест рассматриваемой архитектуры на её производительность?

Наиболее эффективный способ - это увеличение пропускной способности тракта между процессором и памятью. Сюда входит введение много-уровневой памяти, в том числе КЭШ памяти и разделение её на КЭШ-команд и КЭШ-данных, применение многопортовой памяти, использование принципа расслоения при модульной организации памяти, применение механизмов реализации виртуальной памяти.

Что касается линейного адресного пространства, то его преобразование в структурированное пространство памяти было предложено г. Илиффом в 1972 г. Его идея состоит в том, чтобы с помощью некоторого алгоритма добавлять ко всем элементам памяти информацию, показывающую атрибут этого элемента. Эта дополнительная информация получила название *tag* (тег-признак). Машины, основанные на использовании этого признака, называются *теговыми машинами*. В таких машинах различение типов данных поддерживается аппаратно, так как каждый тип данных характеризуется своим тегом. Это в свою очередь приводит к тому, что однотипные команды, отличающиеся только типом операндов, никак не

различаются. Следовательно, число команд в системе команд таких машин сокращается.

Примером машины со структурированной памятью является SWARD-машина, предложенная г. Майерсом. В SWARD-машине данные представлены структурными элементами, называемыми ячейками, которые состоят из поля тега и поля данных. Память разбита на 4-разрядные единицы. Ниже на рис. 1.3 приведены различные типы ячеек.

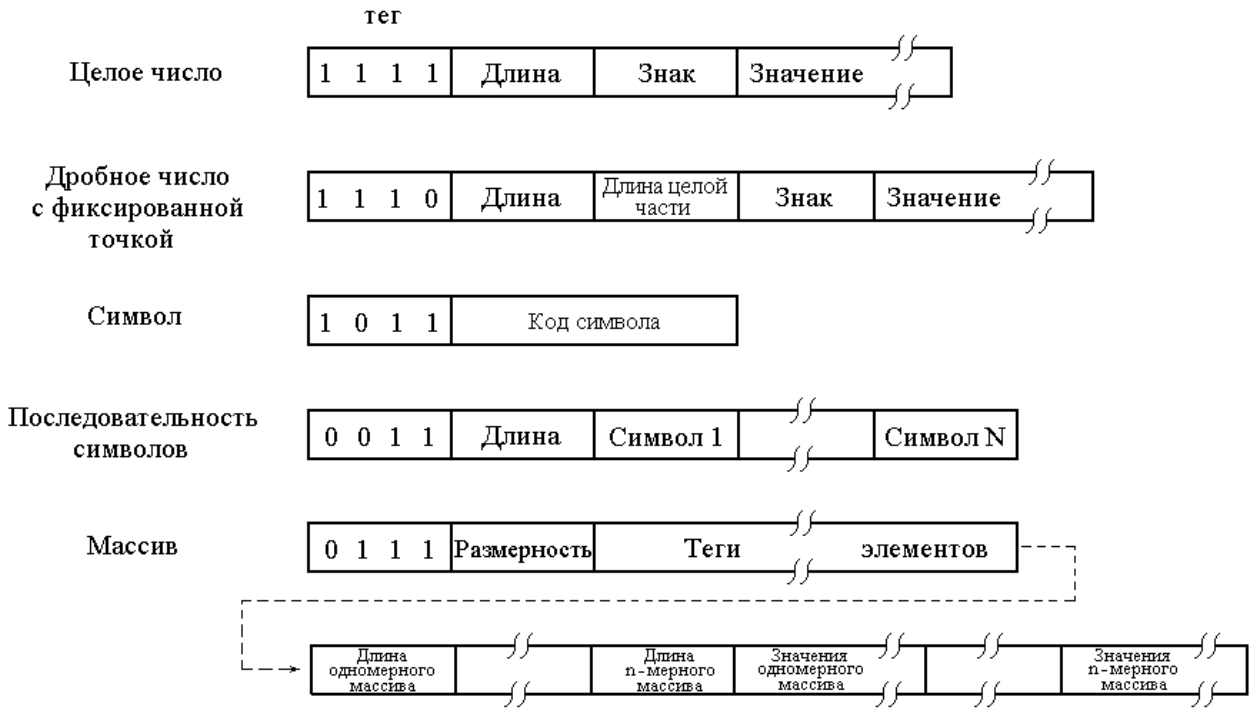


Рис.1.3 Типы ячеек SWARD-машины

Таким образом, применение тегов в том числе устраняет "отсутствие различий в семантике данных" на уровне их машинного представления.

Для улучшения производительности при использовании конвейера вводится блок предсказания ветвлений. Данный блок аппаратно подсчитывает число итераций для арифметических циклов и определяет момент следующего перехода. В случае перехода по признаку возможно лишь вероятностное предсказание. При этом собирается статистика предыдущих переходов.

Устранение побочного эффекта более сложная проблема, решение которой основано на создании архитектуры машины, реализующей так называемый функциональный язык (ФЯ). ФЯ предполагает отсутствие побочного эффекта (отсутствует оператор присваивания) и не влечёт за собой последовательное считывание и исполнение команды. Примерами ФЯ являются языки LISP и SALS. Программа состоит из множества равенств,

определяющих функции и описаний приложений этих функций при вводе данных. Например, в языке SALS функция fact, вычисляющая факториал, определяется следующим образом:

fact n where fact 0 = 1
*fact n = n*fact(n-1)*

Следовательно, программа на ФЯ представляется некоторой функцией *F*, а выполнение программы определяется как результат действия этой функции (её оценка) *F(x)* по отношению к входу *x*.

Лекция 3

2. Эволюция способов обработки данных

Обычно в учебной литературе, рассматривая эволюцию средств вычислительной техники, приводят критерии разбиения ЭВМ на поколения и их подробные характеристики. Поскольку данный курс лекций рассчитан на специалистов в области вычислительной техники, способных вести разработку аппаратных средств обработки данных, обратим более пристальное внимание на эволюцию способов обработки данных и механизмов их обеспечивающих.

Разделим все способы обработки данных на два класса - *скалярная обработка и векторная обработка*. Исторически всё начиналось со скалярной обработки, которая реализовывалась с помощью базового варианта неймановской архитектуры. Причём первые ЭВМ имели в своём составе арифметическое устройство (АУ) последовательного действия. Например, Малая электронная счётная машина (МЭСМ), созданная под руководством академика С.А. Лебедева в 1950 г., имела производительность 50 оп/с, последовательное АУ с фиксированной запятой, всего 4 арифметические операции и одну управления при ёмкости оперативного запоминающего устройства (ОЗУ) в тридцать одно слово. В 1953 г. была создана Большая электронная счётная машина (БЭСМ), которая обладала производительностью 12000 оп/с, имела параллельное 39-разрядное АУ с плавающей запятой, ёмкость ОЗУ 1024 слова и выполняла 32 различные операции. БЭСМ имела достаточно развитую систему внешней памяти - барабан, ленту, устройства ввода/вывода на перфокартах и печатающее устройство. Согласно жёсткой последовательности работы устройств БЭСМ (см. рис. 2.1 ниже) сначала из ОЗУ считывался код команды, в соответствии с которой устройство управления организовывало необходимые обращения к ОЗУ за считыванием первого и второго операнда (скаляра), выдавало управляющие сигналы для выполнения операции на АУ, затем результат операции записывался в ОЗУ. Далее считывалась следующая команда. При обращении к ленте, барабану или внешним устройствам, устройство управления обеспечивало все операции поиска информации, её считывания и записи с применением АУ.

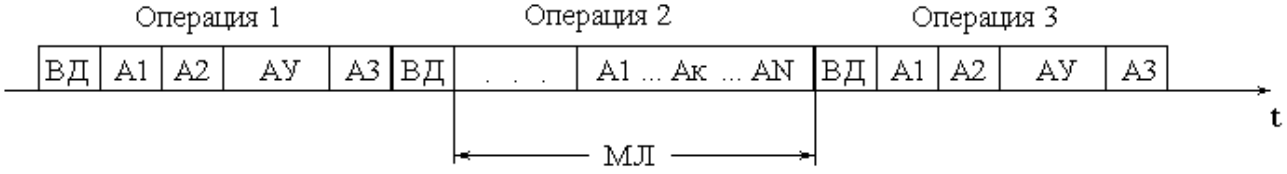


Рис 2.1 Временная диаграмма работы устройств БЭСМ

На приведённой временной диаграмме введены обозначения:

- ВД - выборка и дешифрация команды;
- А1 - выборка первого операнда;
- А2 - выборка второго операнда;
- АУ - выполнение соответствующей операции на АУ;
- А3 - запись результата операции в ОЗУ;
- МЛ - операция записи и считывания с магнитной ленты.

Следующим шагом в развитии скалярной обработки было введение *механизма предварительного просмотра команд*. Это позволило совместить такие этапы выполнения команды как её выборка и декодирование с исполнением предыдущей команды. Примерами машин с данным усовершенствованием служат ЭВМ М-40 и М-20, разработанные в Институте точной механики и вычислительной техники (ИТМ и ВТ) в 1953-58 годах. М-40 достигала производительности 40000 оп/с при ёмкости ОЗУ 4096 40-разрядных слов. Для достижения столь высокой производительности были существенно пересмотрены принципы организации системы управления ЭВМ. Каждое устройство машины получило своё автономное устройство управления, что позволило реализовать их параллельную работу. На рис. 2.2 ниже представлена временная диаграмма выполнения последовательности команд на ЭВМ М-40.

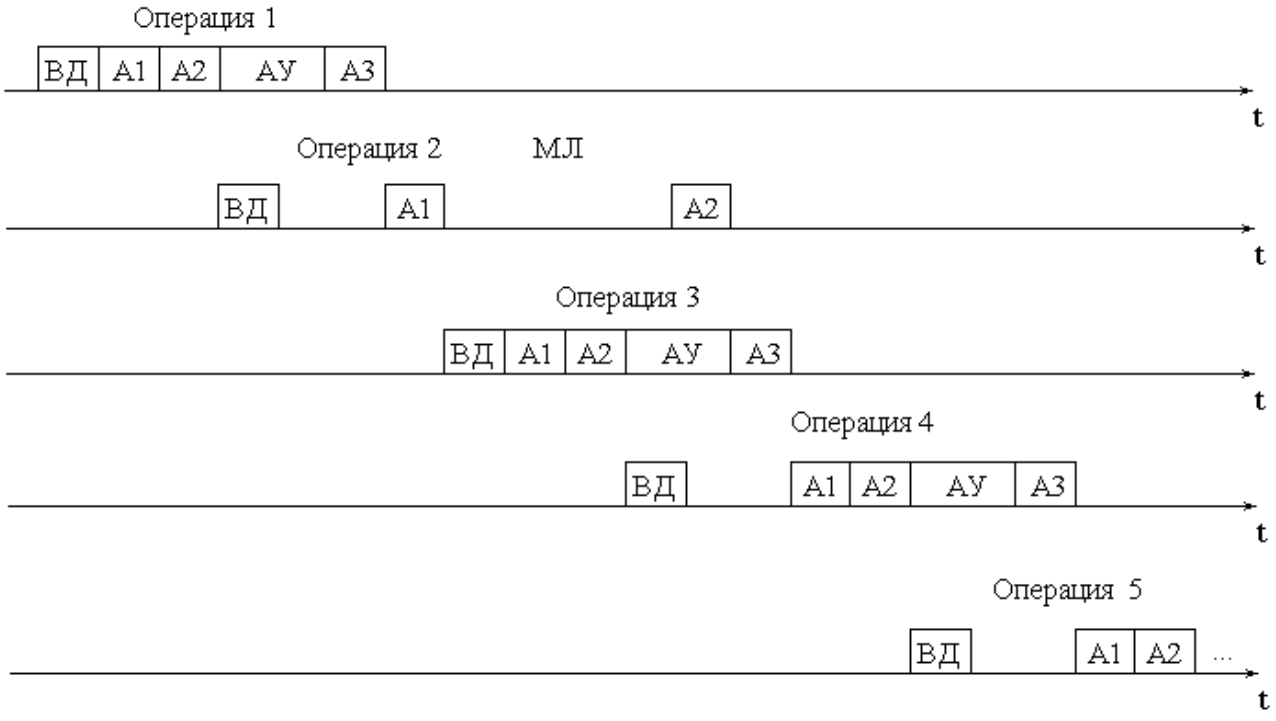


Рис. 2.2 Временная диаграмма выполнения последовательности команд на ЭВМ М-40

Как только была доказана высокая эффективность параллельной работы отдельных устройств ЭВМ, появилась идея введения множества автономных исполнительных функциональных блоков, каждый из которых предназначен для выполнения заданных операций. Таким образом, возник *функциональный параллелизм* на уровне операций. Примером ЭВМ, в которой применён так называемый параллельный процессор (многофункциональный процессорный модуль, содержащий несколько исполнительных функциональных блоков), может служить машина CDC-6600 (1964 г.). Она содержала 10 операционных исполнительных блоков - сумматор с плавающей точкой, умножитель с фиксированной точкой, сумматор с фиксированной точкой и др., которые параллельно выполняют соответствующие операции, совместно используя одни и те же регистры под управлением одного и того же устройства управления (УУ). УУ поддерживает в определенном состоянии операционные блоки (ОБ) и регистры, в процессе выполнения команд производит проверку занятости необходимых ОБ, управляет поступлением требуемых данных из других ОБ и памяти и передаёт управление определённым ОБ, когда для выполнения ими команд всё готово. При невозможности выполнения команды из-за того, что данные не готовы, УУ заставляет ждать ОБ, пока выполнение команды будет возможным.

Таким образом, если длительность операции, задаваемой последующей командой, меньше и связь между её данными и результатами выполнения предыдущей команды отсутствует, то результат этой операции окажется в регистре раньше, чем результат операции предыдущей команды. Так в CDC-6600 новая команда поступает в каждом машинном цикле, тогда как быстродействие ОБ позволяет произвести суммирование с фиксированной точкой за три машинных цикла, умножение с плавающей точкой - за 10, а деление с плавающей точкой за 29 машинных циклов. Таким образом, для того, чтобы темп поступления команд не нарушался, должно быть предусмотрено определённое число операционных блоков соответствующих типов. Очевидно, что для различных прикладных задач, содержащих различное сочетание команд (в том числе и различный процент встречаемости разных команд в программе), эффективность применения параллельного процессора будет сильно варьироваться от задачи к задаче. Следует отметить, что принцип управления вычислительным процессом, используемый в CDC-6600, является одним из прообразов *принципов управления в потоковых машинах с управлением данными*. В таких машинах команды, для которых готовы все данные (операнды), в принципе, могут выполняться независимо от их места в программе. С другой стороны, параллельный процессор CDC-6600 является родоначальником самой современной идеологии построения *процессоров с очень длинным командным словом.*

Реализация параллельного процессора в CDC-6600 не была бы столь эффективной, если бы в нём не был применён, хоть и в зачаточном виде,

конвейер команд. Поэтому рассмотрим следующий способ усовершенствования скалярной обработки - конвейер команд. Конвейер команд реализует способ обработки неявных векторов. Сущность его состоит в следующем. (Следует обратить внимание на то, что конвейер команд является развитием способа предварительного просмотра команд). Последовательность выполнения каждой команды разделяется в процессоре на основные этапы:

а) выборка команды (B);
б) декодирование команды (D);
в) вычисление адреса операнда (A);
г) выборка операнда (F);
д) выполнение операции (E);
е) запоминание результата (Z);

Приведённое разделение является условным, так как в каждом конкретном процессоре и число этапов, и сами этапы могут быть различными, но сама сущность последовательности выполнения команды остаётся неизменной (неймановская архитектура). Наиболее общий приём конвейеризации заключается в том, что для выполнения соответствующей этапу операции вводятся специализированные исполнительные узлы, и при этом, когда в конвейере заканчивается исполнение определенного этапа предыдущей команды, высвобождается соответствующий исполнительный узел и может быть начато выполнение аналогичного этапа следующей команды. В идеальной ситуации очередная команда должна поступать на конвейер в тот момент, когда информация предыдущей команды ушла из первого узла во второй, что определяет, в нашем случае, шестикратное повышение производительности, в случае, если все узлы выполняют свои операции за одинаковое время $t_{эт}$ (этапа). Тогда каждая команда требует время $6t_{эт}$ для своего исполнения. На выходе конвейера результаты выполнения каждой из последовательности команд записываются в память через $t_{эт}$. Конвейер команд условно показан на рис. 2.3 ниже.

Рис. 2.3 Конвейер команд

$T_{латентное}$ – время от запуска конвейера до выдачи первого результата;
$1/T_{ср}$ – интенсивности выходного потока конвейера;
$T_{ср}$ – среднее время появления результатов работы конвейера.

В табл. ниже приведены данные последовательной реализации этапов выполнения *i*-ой команды. При этом строки таблицы отмечены названиями

устройств или узлов, выполняющих соответствующий этап выполнения команды. Так, этапы а), г) и е) реализуются с обращением к памяти, этап б) - в устройстве управления (УУ), этап в) - в блоке вычисления адреса (БВА), д) - в АЛУ.

Таблица.2.1 Последовательная реализация этапов выполнения i-ой команды

Узлы	**Последовательность прохождения команды по исполнительным узлам во времени**					
Память	А).Выборка i-ой команды из памяти	i+1	i+2	i+3	i+4	i+5
УУ	i-1	б).Декодиро-вание i-ой команды	i+1	i+2	i+3	i+4
БВА	i-2	i-1	в).Вычисление адреса операнда i-ой команды	i+1	i+2	i+3
Память	i-3	i-2	i-1	г).Выборка операнда i-ой команды	i+1	i+2
АЛУ	i-4	i-3	i-2	i-1	д).Выполнение i-ой команды	i+1
Память	i-5	i-4	i-3	i-2	i-1	е).Запоминание результата i-ой команды

Из рассмотрения данной таблицы видно, что при выполнении одной команды происходит три обращения к памяти. Причём для разных команд число обращений к памяти может варьироваться. Кроме того, если команды не выполняются одна за другой или не в каждой команде используются все исполнительные узлы конвейера, а также если этапы выполнения команды имеют различную продолжительность, оптимальное повышение производительности не достигается. Нарушения последовательности событий имеют место либо при выполнении команд переходов и обращения к подпрограммам, либо при переключениях контекста. В этих случаях конвейер нарушается. Конвейер нарушается и в том случае, когда результат *i*-ой команды используется как операнд *(i+1)*-ой. При выполнении коротких команд некоторые узлы конвейера не используются. К тому же, одни из них работают медленнее, чем другие. Эти обстоятельства также снижают общую производительность конвейера. Подробно этот вопрос рассмотрен потому, что на определённом этапе развития вычислительной техники стали видны

проблемы, которые необходимо решать для улучшения скалярной обработки данных.

Для грубой оценки повышения производительности процессора, имеющего место в результате применения конвейера, можно принять, что самым медленным участком конвейера является узел, в котором собственно происходит выполнение операции, предписанной командой. Тогда повышение производительности в конвейере определяется в первом приближении отношением суммы средних значений времени работы всех узлов конвейера к средней величине времени работы самого медленного участка конвейера, при этом среднее берётся для заданной комбинации команд. В любом конвейере производительность определяется самым длинным участком.

Нарушение строго последовательного выполнения команд программы вызывает необходимость очистки конвейера от команд, выполнение которых началось после команды, нарушившей эту последовательность, и повторного заполнения конвейера. Доля команд, на которых естественная последовательность выполнения программы нарушается, обычно составляет 15-20% их общего количества, а вызываемое процентное снижение производительности превышает вероятность их появления в программе. Таким образом, из приведённых рассуждений видно, что основными проблемами при организации конвейера являются следующие:

- первая проблема - это уменьшение числа обращений к памяти при выполнении одной команды;
- вторая - это решение задачи разбиения цикла исполнения команды на равные по времени этапы;
- третья - это разработка механизмов, предотвращающих большие потери производительности процессора при нарушении последовательности выполнения команд.

Первая проблема в современных процессорах решается за счёт введения модульной организации памяти с применением принципа расслоения памяти, построением иерархической сверхоперативной памяти (КЭШ 1-го и 2-го уровня), выполняющей функцию буфера между процессором и оперативной памятью, разделением КЭШа на КЭШ команд и КЭШ данных. Одним из путей преодоления второй проблемы может быть уменьшение времени этапа д) собственно выполнения команды, за счёт его внутренней конвейеризации. В качестве примера рассмотрим команду умножения с плавающей точкой двух десятичных чисел 3,8X10^2 и 9,6X10^3. Процессор выполняет эту операцию за три шага:

- перемножает мантиссы (3,8X9,6=36,48);
- складывает порядки (2+3=5);
- нормализует результат, т.е. размещает в необходимом месте десятичную запятую (3,648X10^6).

Если узел, выполняющий операцию умножения с плавающей точкой, разбить на три самостоятельных узла, каждый из которых будет предназначен для выполнения одного их трёх шагов, то во время выполнения шага 3, узлы, выполняющие шаги 1 и 2 не заняты никакими действиями. Применение конвейера могло бы позволить исполнительным узлам 1 и 2 обрабатывать следующую пару чисел одновременно с выполнением шага 3 над данной парой. Данный пример характерен ещё и тем, что иллюстрирует *параллелизм на уровне операций* в команде, т.е. шаги 1 и 2 для одной команды могут выполняться параллельно на самостоятельных исполнительных узлах. То, в какой мере конвейер может ускорить работу процессора, зависит от структуры конвейера. Организация конвейера может быть предназначена и для таких операций, как пересылка данных и их извлечение из памяти, и для арифметических операций. Если какая-то операция требует сразу два или более операндов, находящихся в памяти и извлечение операнда из памяти занимает несколько тактов, то при конвейерной организации процессор может начинать циклы обращения к элементам данных на каждом такте.

Решение третьей проблемы возможно с помощью таких приёмов, как, например, определение наиболее вероятного изменения последовательности команд при выполнении команд перехода или выполнения обеих ветвей программы. И то и другое реализуется с помощью введения специальных аппаратных средств, например, так называемого блока ветвлений и организации второго конвейера команд. Процессор с *двумя и более конвейерами команд* называется *суперскалярным процессором.*

Таким образом, мы рассмотрели этап развития скалярной обработки, который определяет обработку неявных векторов, т.е. таких, которые не связаны ни с введением в состав системы команд векторных команд, ни с введением в структуру процессора специальных схем поддержки обработки векторных данных. Этот этап завершает длительный процесс формирования структуры процессора, организация которого ориентирована на выборку операндов из памяти и занесение результатов операций в память. Такие структуры получили название *"память-память"*.

Промежуточным вариантом между СОД, обладающими структурой "память-память" и структурой *"регистр-регистр"*, является СОД, включающая в свой состав специализированные процессоры. К ним относятся матричные, векторные и ассоциативные процессоры. Суть их заключается в том, что все они предназначены для обработки явных векторов. Более подробно эти процессоры будут рассмотрены позднее. Здесь следует только отметить, что типичными представителями структуры "регистр-регистр" являются так называемые RISC (Reduced Instruction Set Computer) процессоры, обладающие таким сокращённым набором команд, что этап исполнения каждой из них в конвейере команд равен одному такту. В структуру таких процессоров введено большое количество регистров,

позволяющее исполнительным устройствам АЛУ работать только с регистрами.

Особенное место среди процессоров со структурой, близкой к структуре "регистр-регистр", но взявшей лучшие черты и от структуры "память-память", является процессор с очень длинным командным словом (VLIW – Very Long Instruction Word). Этот тип процессоров будет также рассмотрен позднее. Но здесь необходимо отметить следующее. Механизм, обеспечивающий реализацию VLIW, основан на *синхронном параллельном выполнении нескольких команд одной программы.*

Следующим усовершенствованием обработки данных явилось введение *асинхронного параллельного выполнения независимых ветвей одной программы или разных программ* за счёт построения многопроцессорной вычислительной системы (МВС) с децентрализованным управлением вычислительным процессом. Собственно изучению МВС и посвящены основные разделы данного курса лекций. Дальнейшее развитие способов обработки данных, на мой взгляд, будет связано с появлением новых архитектурных решений, связанных как с усовершенствованием неймановской архитектуры, так и построением новых, например, такой, о которой заявил российский учёный Б.А. Бабаян.

Ниже приведена обобщённая схема эволюции способов обработки данных (см. таблицу ниже).

Таблица. 2.2 Эволюция способов обработки данных

Вновь вводимые способы обработки данных	**Механизмы их реализации**
Базовый вариант скалярной обработки	Централизованное управление последовательным процессом исполнения команд программы
Предварительный просмотр команд	Параллельное выполнение выборки и декодирования команды с исполнением предыдущей команды
Функциональный параллелизм	Параллельная работа нескольких автономных исполнительных функциональных узлов
Обработка неявных векторов	Конвейер команд
Суперскалярная обработка	Несколько конвейеров команд
Обработка явных векторов	Централизованное управление параллельным процессом исполнения векторных команд в специализированных процессорах
Синхронное параллельное выполнение нескольких команд одной программы	Очень длинное командное слово
Асинхронное параллельное выполнение независимых ветвей одной программы или разных программ	Децентрализованное управление множеством процессоров

Лекция 4

3. Уровни параллелизма при выполнении прикладных задач

Как следует из рассмотрения предыдущего материала, повышение производительности ВС базируется на идее распараллеливания вычислительных процессов [2]. При этом реализация параллельных процессов потребовала создания не только аппаратных средств многопроцессорных вычислительных машин и систем, но и соответствующего программного обеспечения: языковых средств описания параллельных процессов, соответствующих трансляторов, операционных систем и прикладных программ, о которых речь пойдёт позднее. Но все методы организации параллельных вычислений и способы их реализации основаны на существовании двух парадигм, имеющих отношение к автоматической обработке данных. Это *параллелизм данных* и *параллелизм прикладных задач*. В англоязычной литературе соответствующие термины обозначаются как data parallel и message passing. В основе использования свойств объектов обработки данных, определяемых этими парадигмами, лежит распределение вычислительной работы по имеющимся ресурсам вычислительной системы так, чтобы минимизировать время выполнения поступивших на исполнение задач.

Параллелизм данных имеет место тогда, когда по одной и той же программе должна обрабатываться некоторая совокупность данных, поступающих в систему от нескольких источников одновременно. Например, обработка информации от датчиков, измеряющих одновременно один и тот же параметр и установленных на нескольких однотипных объектах. Основная идея подхода, основанного на параллелизме данных, заключается в том, что одна операция может выполняться сразу над всеми элементами массива данных. Следовательно, различные фрагменты такого массива должны обрабатываться на различных исполнительных узлах системы.

Параллелизм прикладных задач имеет место тогда, когда прикладная задача может быть разбита на несколько относительно самостоятельных подзадач, каждая из которых может выполняться на разных процессорах параллельно над множеством, в общем случае, различных и разнотипных данных.

Рассуждая о распараллеливании вычислительных процессов, необходимо иметь в виду, что, прежде всего, должна быть возможность совмещения во времени каких-либо этапов решения задач. Степень детализации при разделении процесса решения задачи на этапы и определяет уровни параллелизма при выполнении прикладных задач. Здесь следует обратить внимание на следующее. Рассмотренные парадигмы не связаны с организацией конвейеров операций и команд, рассмотренных выше, хотя и тот и другой являются реализацией низших уровней параллелизма. Однако здесь параллелизм связан с совмещением во времени выполнения различных этапов в первом случае следующих друг за другом операций, а во втором команд, сохраняя последовательный характер обработки данных. Что же касается непосредственно распараллеливания вычислительного процесса, то можно выделить следующие уровни параллелизма. Первый - это *уровень скалярных операций*,

когда различные операции одновременно выполняются на различных исполнительных узлах над различными данными. Пример:

$X_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$; три операции ($b^2$, 4ac, 2a) можно выполнить параллельно.

Примером процессора с таким видом параллелизма является процессор машины CDC-6600, рассмотренной выше. Второй - это *уровень векторных операций*, когда одна и та же операция одновременно выполняется на одинаковых исполнительных узлах над различными однотипными данными. Пример: усреднение яркости точек изображения (усреднение яркости двух соседних точек → усреднение яркости столбца → усреднение яркости столбцов). Уровень векторных операций причислен к более высокому уровню потому, что изначально в одной векторной команде предусматривается выполнение множества одинаковых операций. Процессоры, предназначенные для реализации такого уровня параллелизма, называются *векторными процессорами*. Следует отметить, что для удешевления построения векторного процессора выполнение векторных операций, инициированных одной векторной командой, может происходить на одном исполнительном узле, реализующим принцип конвейера операций. Примером таких процессоров являются процессоры фирмы CRAY Research. Процессоры, в которых введены по четыре одинаковых исполнительных узла, способных одновременно выполнять по четыре одинаковых операций одной векторной команды, применены фирмой NEC в суперЭВМ семейства SX.

Одним из наиболее распространённых типов параллелизма в обработке информации является параллелизм на *уровне независимых ветвей (подзадач) прикладной задачи*. Прежде чем подробнее остановиться на этом уровне параллелизма, отметим, что существует более высокий уровень - *уровень естественного параллелизма независимых задач*. Реализация этого уровня базируется на предположении о том, что в ВС поступает непрерывный поток не связанных между собой задач, решение которых не зависит от результатов решения других задач. Наличие же в ВС нескольких модулей обработки данных, например, процессоров, позволяет практически совмещать во времени выполнение таких задач. Поскольку этот уровень не требует более подробных комментариев, отметим, что мы ввели понятия 4-х уровней параллелизма, а теперь вернёмся к пояснению уровня параллелизма независимых ветвей прикладной задачи.

Двумя *независимыми ветвями* одной программы будем считать такие её части, при выполнении которых выполняются следующие условия:

- ни одна из входных для ветви программы величин не является выходной величиной другой ветви программы (отсутствие функциональных связей);
- для обеих ветвей программы не должна производиться запись данных в одни и те же ячейки памяти (отсутствие связи по использованию одних и тех же ячеек оперативной памяти);

- условия выполнения одной ветви не зависят от результатов или признаков, полученных при выполнении другой ветви (независимость по управлению);
- обе ветви не должны использовать одни и те же стандартные подпрограммы (программная независимость).

Следовательно, такие ветви одной программы при наличии в ВС нескольких процессоров могут выполняться параллельно и независимо друг от друга.

Хорошее представление о параллелизме независимых ветвей даёт ярусно-параллельная форма (ЯПФ) представления программы, которая обеспечивает возможность, используя формальное представление с помощью соответствующего языка параллельного программирования, автоматизировать процесс распараллеливания задачи. На рис. 3.1 приведён пример представления части программы в ЯПФ.

Кружками с цифрами внутри обозначены ветви (цифра - порядковый номер ветви). Длина ветви представляется цифрой, стоящей около кружка, и может соответствовать числу временных единиц, которые требуются для её исполнения (например, число машинных тактов). Стрелками показаны входные данные и результаты обработки. Входные данные обозначаются символом X, нижние цифровые индексы которых соответствуют номеру входных величин. Выходные данные обозначаются символом Y, верхний цифровой индекс которых соответствует номеру ветви, при выполнении которой получен данный результат, а нижний индекс означает порядковый номер результата, полученного при реализации данной ветви программы. Изображённая на рисунке программа содержит 9 ветвей, расположенных на трёх ярусах. Очевидно, что для ветвей каждого яруса выполняется первое условие независимости. Допустим, что выполняются и остальные три условия независимости ветвей (на графе для простоты изложения отсутствуют связи по памяти, управлению и подпрограммам). Тогда можно считать, что ветви одного яруса являются независимыми и, следовательно, есть принципиальная возможность при наличии в ВС нескольких процессоров, организовать параллельные вычисления. При этом возникает следующая задача. Анализируя рисунок ниже, можно получить результат – суммарное время выполнения программы на одном процессоре равно 255 единицам времени. Если система содержит n процессоров, то необходимо так распределить ветви программы, чтобы общее время решения программы было оптимальным. При этом предполагается, что любая ветвь может выполняться только тогда, когда для неё готовы все входные данные (следует отметить, что мы здесь не учитываем такой фактор, как объём передаваемых данных от одной ветви к другой, который может внести коррективы в полученный результат распределения ветвей по процессорам).

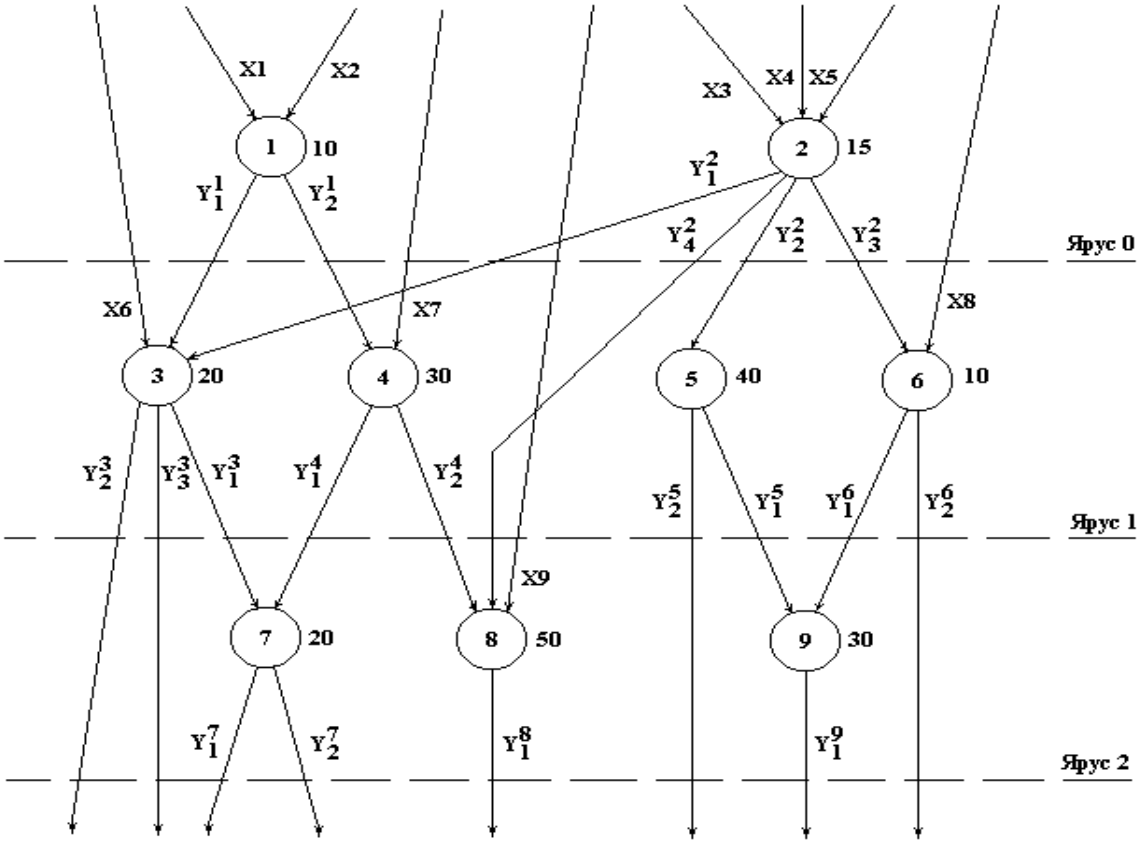


Рис.3.1 Представление программы в ярусно-параллельной форме

Допустим, n=2, тогда одним из решений может быть следующее. Первый процессор выполняет последовательно ветви 1-4-5-9 и тратит на это 110 единиц времени. Второй процессор выполняет 2-3-6-8-7 ветви и тратит 115 единиц времени. Таким образом, все ветви программы выполняются в целом за 115 единиц времени. Очевидно, что решений распределения будет несколько, а эффективность реализации любого распределения определяется в том числе точностью сведений о длине ветви (обычно известна приближённая её оценка) и имеющейся в ВС процедурой автоматического выбора пути решения задачи и слежения за выполнением всех её ветвей. Заметим, что практически никогда не удаётся избежать некоторых простоев процессоров, которые возникают из-за отсутствия исходных данных для выполнения той или иной ветви, а это ведёт к уменьшению производительности ВС с параллельной обработкой на уровне ветвей задачи. Ещё один вывод, который можно сделать из рассмотрения данного материала, состоит в том, что с одной стороны выделить независимые ветви программ при их разработке достаточно сложно, а с другой стороны в языках программирования должны присутствовать специальные средства, отображающие параллелизм независимых ветвей программы.

При рассмотрении параллельной обработки на всех уровнях параллелизма всегда следует иметь в виду, что любая программа по своей сути является совокупностью двух частей - последовательной и параллельной. Поэтому, если, например, программа состоит из последовательной части, со-

держащей S блоков, и параллельной части, содержащей N блоков, причём каждый из последовательных блоков выполняется за время Ti, а параллельные блоки выполняются за одинаковое время Tn, то общее время выполнения программы будет

T =Ts+N*Tn,

где Ts-суммарное время выполнения последовательных частей программы. Тогда возможный уровень параллелизма программы можно определить из выражения

P=N*Tn/(Ts+N*Tn).

Таким образом, если параллельная программа выполняется на M процессорах, то *коэффициент ускорения* параллельной обработки равен

Ky=(Ts+N*Tn)/(Ts+N*Tn/M).

Из этой формулы видно, что даже при M=N, эффективность параллельной обработки определяется долей последовательной части программы.

Какое максимально возможное значение коэффициента ускорения можно получить, выполняя программу на M процессорах, даёт так называемый *закон Амдала*, в соответствии с которым

Ky<1/(f+(1-f)/M),

где f-доля операций в программе, которые нужно выполнять последовательно (при этом доля понимается не по статическому числу строк кода, а по числу операций в процессе выполнения); (1-f)-доля операций в программе, которые можно выполнить параллельно.

Крайние случаи в значениях f соответствуют полностью параллельным (f=0) и полностью последовательным (f=1) программам. Если для конкретной программы, например, f=0.1, то ускорения более, чем в 10 раз получить в принципе невозможно вне зависимости от числа процессоров, реализующих параллельную часть. Следовательно, при формировании алгоритма параллельной программы, необходимо самое серьёзное внимание уделять поиску путей *распараллеливания последовательной части программы*.

4. Распараллеливание последовательных частей программ

Преобразование последовательной части программы в параллельную форму базируется на следующем [6]. Пусть операции алгоритма разбиваются на группы, упорядоченные таким образом, что каждая операция любой группы зависит либо от начальных данных алгоритма, либо от результатов выполнения операций, находящихся в предыдущих группах. Следовательно, все операции одной группы должны быть независимыми и обладать возможностью быть выполненными одновременно на имеющихся в системе функциональных узлах. Представление алгоритма в таком виде и является *параллельной формой алгоритма*. Тогда каждая группа операций в ней называется *ярусом параллельной формы*, число групп - *высотой параллельной формы*, максимальное число операций в ярусе - *шириной параллельной формы*.

Рассмотрим три уровня представления последовательных программ, на которых их распараллеливание даёт существенный выигрыш в производительности вычислительной системы. (Будем считать, что уровень независимых ветвей задачи с одной стороны рассмотрен выше, а с другой стороны он, в большинстве случаев, и определяет параллельную часть программы). Итак, первый уровень основан на том, что программа разбивается на относительно небольшие участки, достаточно тесно связанные между собой как по данным так и по управлению, и среди них выделяются те, которые могут быть выполнены параллельно. Реализации таких участков на ВС называются *процессами.* Для определения таких процессов строится граф потока данных, анализируются множества входных (считываемых) переменных R(read) и выходных (записываемых) переменных W(write) каждого процесса. Два процесса i и j могут выполняться параллельно при следующих условиях:

$Ri \wedge Wj = 0$

$Wi \wedge Rj = 0$

$Wj \wedge Wi = 0$

Это означает, что входные данные одного процесса не должны модифицироваться другим процессом и никакие два процесса не должны модифицировать общие переменные.

Второй уровень связан с распараллеливанием циклов. Границы циклов находятся по формальным признакам описания циклов в соответствующих языках программирования. Процесс преобразования последовательного цикла для параллельного выполнения заключается в том, что из переменных формируются векторы, над которыми выполняются векторные операции. Поскольку векторные операции выполняются параллельно над всеми или частью элементов вектора (в зависимости от вычислительных ресурсов ВС), то происходит значительное ускорение вычислительного процесса.

Третий уровень - это распараллеливание линейных участков. *Линейным участком* программы назовём часть программы, операторы которой выполняются в естественном порядке или порядке, определённом командами безусловных переходов. Линейный участок ограничен начальным и конечным операторами.

Начальным оператором линейного участка назовём оператор, для которого выполняется по крайней мере одно из следующих условий:

- у оператора не один непосредственный предшественник;
- у непосредственного предшественника более одного непосредственного последователя.

Конечным оператором линейного участка назовём оператор, для которого выполняется по крайней мере одно из следующих условий:

- у оператора не один непосредственный последователь;
- у непосредственного последователя оператора больше одного непосредственного предшественника.

Исходя из этих определений, легко построить алгоритм нахождения линейных участков программы. Внутри линейного участка распараллеливание может производиться по операторам, а внутри операторов - распараллеливание арифметических выражений. Распараллеливание по операторам позволяет при наличии вычислительных ресурсов выполнять сразу несколько операторов.

Рассмотрим несколько примеров распараллеливания линейных участков:

1. Уменьшение высоты арифметических выражений:

Пусть необходимо вычислить произведение восьми чисел - a1, a2, a3, a4, a5, a6, a7, a8. Обычная схема, реализующая процесс последовательного умножения, выглядит следующим образом:

ярус 1 a1a2
ярус 2 (a1a2)a3
ярус 3 (a1a2a3)a4
ярус 4 (a1a2a3a4)a5
ярус 5 (a1a2a3a4a5)a6
ярус 6 (a1a2a3a4a5a6)a7
ярус 7 (a1a2a3a4a5a6a7)a8

Характеристики этого дерева: высота - 7, ширина - 1. В процессе реализации сложного выражения, возможны преобразования, дающие различную параллельную сложность. Среди них нужно найти алгоритмы с наименьшей высотой. Это главная проблема этих преобразований. Если процессоров несколько, то можно предложить такой вариант уменьшения высоты приведенного дерева:

ярус 1 a1a2 a3a4 a5a6 a7a8
ярус 2 (a1a2)(a3a4) (a5a6)(a7a8)
ярус 3 (a1a2a3a4)(a5a6a7a8)

Характеристики этого дерева: высота - 3, ширина - 4. Таким образом, при использовании четырёх процессоров высота дерева уменьшилась на 4 яруса и, следовательно, время выполнения алгоритма сократилось более чем в два раза.

В общем случае при произвольном числе сомножителей n, высота параллельной формы равна целой части log n. Эта параллельная форма реализуется на n/2 процессорах, но в ней загруженность процессоров уменьшается от яруса к ярусу. Рассмотренный процесс перемножения чисел называется процессом сдваивания (схема сдваивания).

При вычислении произведения n чисел часто требуется знать не только конечный результат, но и все частичные произведения. В этом случае иногда авторы (Трахтенгерц Э.А.) предлагают следующую процедуру вычислений:

ярус 1 a1a2 a3a4 a5a6 a7a8
ярус 2 (a1a2)a3 (a1a2)(a3a4) (a5a6)a7 (a5a6)(a7a8)
ярус 3 (a1a2a3a4)a5 (a1a2a3a4)(a5a6) (a1a2a3a4)(a5a6a7) (a1a2a3a4)(a5a6a7a8)

Как легко заметить, у этого дерева: высота - 3, ширина - 4, а все процессоры на всех ярусах полностью заняты. Однако, следует обратить внимание на то, что на втором и третьем ярусах различные процессоры используют одни и те же результаты предыдущего яруса. Это означает, что данные одновременно должны быть переданы с одного процессора нескольким другим процессорам при реализации алгоритма на ВС с распределённой памятью. При реализации алгоритма на ВС с общей памятью, необходимо сначала результаты вычислений записать в общую память, а затем последовательно считать их процессорами, которые в них нуждаются. Следовательно, с одной стороны число ярусов дерева остаётся неизменным - 3, но время на выполнение ярусов увеличивается.

Следует отметить ещё один момент. В силу ассоциативности операций и последовательный, и параллельный алгоритмы дают одинаковые результаты только в условиях точных вычислений, но они будут давать разные результаты в условиях влияния ошибок округления. В этом случае считается, что устойчивость параллельных алгоритмов при большом числе процессоров оказывается хуже устойчивости последовательных алгоритмов.

2. Преобразование линейных рекурентных выражений:

Рассмотрим блок операторов:

X = BCD + E

Y = AX

Z = X + FG

При использовании одного процессора этот блок может быть вычислен за 6 шагов, если один временной шаг отводится для каждой арифметической операции. Путём замены операторов можно получить следующий блок:

X = BCD + E

Y = ABCD + AE

Z = BCD + E + FG

Таким образом, если ВС обладает множеством свободных процессоров, таковым, что первый оператор выполняется на автономном процессоре и тогда ему необходимо 3 шага, второй и третий операторы выполняются каждый на двух, параллельно действующих процессорах, и тогда они также выполняются каждый за 3 шага. Следовательно, весь блок операторов выполнится за 3 шага на пяти процессорах с коэффициентом ускорения равным 2.

3. Алгоритм компактного размещения данных:

Допустим необходимо L разрядов из каждого элемента массива A(i) плотно упаковать по элементам массива B(j). Элементы массивов представлены 64-разрядными словами, а L<64. Если очередная секция из L разрядов целиком не помещается в очередном элементе B(j), то оставшаяся часть должна быть перенесена в следующий элемент B(j+1). Последовательный алгоритм упаковки очевиден. Параллельный же можно представить следующим образом. Вычисляется периодичность секций из L разрядов, полностью

заполняющих все разряды последовательных элементов массива B(j) и определяется число этих заполненных элементов. Например, L=48, тогда каждые 4 секции из L разрядов будут без остатка заполнять 3 элемента массива B(j). В этом случае каждые 4 секции можно параллельно размещать в последовательные 3 элемента массива B(j).

Лекция 5

5. Оценка производительности ВС

В начале проектирования любой сложной системы разработчики пытаются выделить основное свойство системы, которое необходимо обеспечить в результате её создания. Таким свойством является полное соответствие системы своему назначению.

Степень соответствия системы своему назначению называется *эффективностью* системы. Обычно эффективность не удаётся определить одной величиной, и поэтому её представляют набором величин, называемых характеристиками системы. Естественно, этот набор должен наиболее полно отображать основное свойство системы. Если это свойство мы определим, как способность системы выполнять заданные функции, сохраняя во времени значения своих параметров в заданных пределах, то основными характеристиками системы должны быть следующие:

- производительность
- надёжность
- стоимость

Для того чтобы характеристику, а, следовательно, и эффективность системы оценить количественно, вводится понятие *показателя характеристики*. С точки зрения оценки эффективности вычислительных систем, её показатели определяются параметрами системы, в качестве которых выступают величины определяющие, например, число процессоров, ёмкость памяти, быстродействие основных компонент ВС.

Таким образом, последовательность объектов, с помощью которых осуществляется оценка эффективности ВС, может быть представлена следующим образом:

свойство – характеристика – показатель – параметр

Как было уже определено выше, производительность это характеристика вычислительной мощности системы, определяющая количество вычислительной работы, выполняемой системой в единицу времени (другими словами, количество прикладных задач, выполненных системой в единицу времени). В настоящее время существует большое количество различных способов, методов и методик оценки производительности ВС и это связано не только с существованием огромного количества видов ВС, но и с тем, что производительность определяется большим числом параметров, как самой вычислительной системы, так и прикладных задач. Поэтому существует довольно сложная проблема выделения частных показателей производительности ВС и параметров, оказывающих наиболее сильное влияние на значения этих показателей.

Среди частных показателей можно отметить следующие:

- *номинальная производительность*
- *комплексная производительность*

- *системная производительность*.

Номинальную производительность (НП) ВС можно определить как совокупность таких параметров системы как быстродействия устройств, входящих в её состав (процессора, памяти, канала обмена):

$$НП = \{Б_{ПР}, Б_{П}, Б_{К}\}, \text{ где}$$

$Б_{ПР}$ - быстродействие процессора

$Б_{П}$ - быстродействие памяти

$Б_{К}$ - быстродействие системы коммутации

Например, если $Б_{ПР}$=2ГГц, $Б_{К}$=133МГц, то Pentium 4 будет работать как Pentium II.

Таким образом, номинальная производительность характеризует только потенциальные возможности вычислительной системы, которые не могут быть достигнуты в реальной обстановке решения конкретной прикладной задачи. Следует отметить, что довольно часто номинальная производительность оценивается только одним параметром - быстродействием основного элемента системы - процессора, а в случае нескольких процессоров - суммарным их быстродействием.

Для того, чтобы оценить влияние на производительность таких факторов как конфликты при обращении процессора к памяти, каналам ввода-вывода и другим общим ресурсам системы, используется другой показатель - *комплексная производительность (КП)*. Комплексная производительность, так же как и НП определяется совокупностью тех же параметров, однако, их значения вычисляются как произведения значений быстродействий в формуле для НП на некоторые коэффициенты Ki, принимающие значения меньшие единицы:

$$КП = \{Б_{ПР} \cdot K_1, Б_{П} \cdot K_2, Б_{К} \cdot K_3\}, \text{ где}$$

K_1 - коэффициент, указывающий на простои процессора из-за конфликтов при обращении к разделяемым ресурсам,

K_2 - коэффициент, указывающий на простои памяти, например, из-за занятости канала связи при обращении к ней процессора,

K_3 - коэффициент, указывающий на то, что по каналу связи кроме основной информации передаётся и так называемая служебная информация, например, контрольные биты, которые «съедают» часть быстродействия системы коммутации.

$K_1, K_2, K_3 < 1$, т.е. реально на прикладных задачах максимальное быстродействие не достигается, т.к. процессор иногда простаивает из-за памяти; канал обмена передает не только данные, но и служебную информацию.

При оценке *системной производительности (СП)* необходимо ещё учитывать и то, что на тех же ресурсах, которые заняты для выполнения прикладных задач, реализуются компоненты системного программного обеспечения. Наиболее точной оценкой СП является статистическая оценка при выполнении системой на длительном интервале времени случайного

потока задач, приближенного к реальному режиму функционирования вычислительной системы. В этом случае число задач n, выполненных системой за время T, является случайной величиной и тогда:

$$СП = \frac{n}{T}$$

Другим способом определения системной производительности является её вычисление через среднее значение tcp интервала между моментами окончания обработки задач tj, поступающих на вход системы в случайные моменты времени, тогда:

$$СП = \frac{1}{t_{ÑÐ}}$$

СП = {БПР·K1·K1', БП·K2·K2', БК·K3·K3'}, где
K1', K2', K3' < 1, т.к. выполнение функций операционной системы (ОС) тоже занимает ресурсы процессора.

Пусть задачи в систему поступают в случайные моменты времени 1з(момент поступления задачи 1), 2з …, а результаты обработки выдаются в моменты P1з(момент выдачи результата задачи 1), P2з…, как это показано на рис .

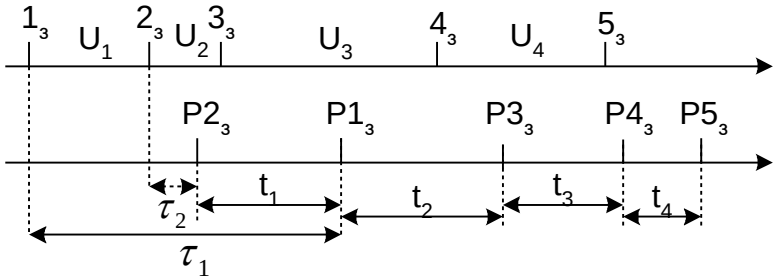


Рис.5.1 Временная диаграмма выполнения потока задач

Анализируя временную диаграмму можно получить

$\lambda = \frac{1}{T_{ÑÐÏ\ 3}}$ - интенсивность входного потока и $T_{ÑÐÏ3} = \frac{\sum_{i=1}^{n-1} Ui}{n-1}$ – среднее время между моментами поступления задач,
где $(n-1)$ – число интервалов, Ui – интервал между моментами поступления задач

Определим τ как интервал времени между моментом поступления задачи и моментом выдачи результата её выполнения, назовём его временем отклика на задание, а через t_{CP} – среднее время выдачи результатов, тогда

$\eta = \frac{1}{t_{ÑÐ}}$ - интенсивность выходного потока (число задач, выполненных системой в единицу времени).
Таким образом, η является оценкой системной производительности (СП).

$$t_{ÑÐ} = \frac{\sum_{i=1}^{n-1} t_i}{n-1}$$

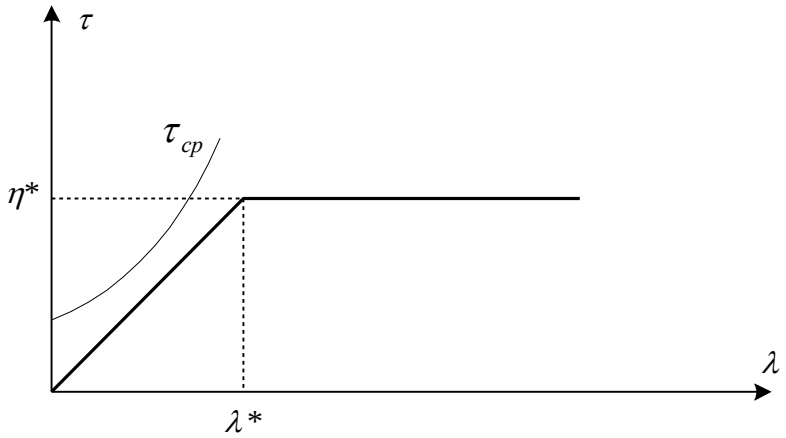


Рис.5.2 Зависимость интенсивности выходного потока и времени отклика от интенсивности входного потока

η^*, λ^* - критические значения, т.е. выше λ^* входной поток делать не имеет смысла.

$\tau_{ÑÐ} = \frac{\sum_{i=1}^{n} \tau_i}{n}$ - среднее время отклика (решения задачи)

При проектировании вычислительных систем по критерию производительности у её разработчиков возникает сложная проблема так организовать вычислительный процесс, чтобы входящие в её состав аппаратные средства были бы максимально загружены.

Показателем эффективного использования этих средств в процессе функционирования системы является коэффициент их загрузки КЗ. Он определяется как отношение времени Ti, в течение которого i-ое рассматриваемое устройство занято выполнением непосредственно своих функций, ко всему времени наблюдения системы Т:

$$КЗ = \frac{T_i}{T}$$

Таким образом, рассмотрены способы оценки производительности через применение по существу тактовой частоты работы основных устройств системы (либо длительности цикла) и статистических методов. Однако, существуют и другие методы оценки производительности вычислительных систем, которые широко применяются на практике как разработчиками так и их пользователями. Рассмотрим их более подробно.

Лекция 6

6. Методы оценки производительности ВС

Способы измерения «комплексной» производительности:

1. Использование смеси команд
2. Аналитический метод
3. Программное моделирование
4. Использование тестовых программ
   - представительных
   - испытательных
   - синтетических

1. Одним из более простых методов является оценка производительности ВС с помощью так называемой смеси команд. Этот метод даёт лучшую оценку по сравнению с оценкой НП, представленной выше, однако тоже характеризует только аппаратную часть системы. Суть метода состоит в том, что для выбранного класса прикладных задач, определяется типовая смесь команд, которая формируется путём анализа частоты встречаемости различных команд при выполнении программ, отображающих решаемые задачи. На основе такого анализа отдельным командам или совокупностям однотипных команд присваиваются весовые коэффициенты. Например, для научно-технических задач определена смесь, названная по имени автора «смесь Гибсона» и, которая выделяет следующие виды команд и их весовые коэффициенты см. таблицу:

Таблица. 5.1

Виды команд и их весовые коэффициенты в смеси Гибсона

Операция	K_i
Сложение, вычитание с фиксированной запятой	0.33
Умножение с фиксированной запятой	0,006
Деление с фиксированной запятой	0,002
Сложение, вычитание с плавающей запятой	0,073
Умножение с плавающей запятой	0,04
Деление с плавающей запятой	0,016
Команды безусловной передачи управления (Jump)	0,175
Команды условной передачи управления (Jxx)	0,065

где, К – коэффициент встречаемости, при использовании смеси команд производительность системы η вычисляется следующим образом:

$$\eta = \frac{\sum_{i=1}^{L} K_i}{\sum_{i=1}^{L} K_i \cdot t_i}, \text{ где}$$

t_i - продолжительность выполнения i-ой команды (например, в машинных циклах)

L - число команд (видов команд) в смеси

Причем, здесь нет учета конфликтов между памятью и процессором.

2. Одним из самых распространённых методов оценки производительности систем в научных кругах является использование аналитических моделей. Метод основан на представлении ВС чаще всего в терминах теории массового обслуживания (ТМО). Математические модели обычно отображают поведение системы в целом с заранее предусмотренными существенными ограничениями. И даже с такими ограничениями не всегда удаётся построить математическую модель системы. В этом случае прибегают к её декомпозиции, создавая математические модели отдельных частей системы с последующим их объединением. Основная проблема, связанная с построением моделей, состоит в том, что каждая такая модель должна разрабатываться индивидуально для очень узкого класса систем, а иногда и только для конкретной системы. К недостаткам этого метода также относятся следующие:

1. Невозможность учитывать функционирование системного программного обеспечения.
2. Невозможность учитывать случайные внешние и внутренние прерывания вычислительного процесса.
3. Трудности изменения набора входных параметров моделей.
4. Сложность оценки адекватности аналитических моделей реальной системе.
5. Сложность выбора математического аппарата и высокая трудоёмкость построения самой модели.

Следует обратить внимание также на то, что высокопроизводительные системы в настоящее время строятся с обеспечением свойства масштабируемости, учитывать некоторые элементы которого в аналитических моделях практически невозможно.

Следующие три метода основаны на использовании тестовых программ. Первый из них реализуется с помощью так называемых *представительных программ*. Такие программы строятся на основе использования типичных для данной области применения систем либо последовательностей команд, либо частей программы, либо целых программ. Представительные программы создаются чаще всего на языке ассемблера, но иногда и на языке высокого уровня, и с их помощью оценивается системная производительность, правда не всегда с учётом реального потока обрабатываемых данных.

Второй метод использует так называемые *испытательные программы*, которые специально создаются для адекватного отображения реального режима функционирования вычислительной системы. Испытание ВС

происходит как в режиме трансляции так и исполнения испытательной программы. Обычно такой метод применяется для оценки производительности специализированных ВС.

Третий метод основан на применении *тестовых программ*, которые являются комбинацией представительных и испытательных программ, и которые получили наименование *синтетических программ*. Преимущество специальной совокупности синтетических программ состоит в том, что она может быть использована для сравнительной оценки ВС одного класса, производимых различными фирмами. Подробно этот вопрос будет рассмотрен ниже.

Наконец нельзя не выделить ещё один метод оценки производительности ВС, который по своим возможностям является одним из самых эффективных методов. Это метод *программного моделирования*. Принципиально с его помощью могут быть учтены все параметры ВС, нюансы её функционирования, при этом, конечно, надо учитывать стоимость создания такой модели. Главным преимуществом программного моделирования является возможность его использования ещё на стадии разработки ВС.

7. Система тестов SPEC. Единицы измерения производительности ВС

SPEC (*Standard Performance Evaluation Corporation*) – это корпорация, созданная в 1988 году, объединяющая ведущих производителей вычислительной техники и программного обеспечения. Основной целью этой организации является разработка и поддержка стандартизованного набора специально подобранных тестовых программ для оценки производительности новейших поколений высокопроизводительных компьютеров [*http:/www.parallel.ru./computers/benchmarks/spec.html*] [С7].

Рассмотрим более подробно применение тестовых программ, для оценки производительности ВС. Однако прежде чем перейти к существу проблемы сделаем отступление для определения единиц измерения производительности, которыми пользуются создатели тестов.

Сначала обратим внимание на следующее. Номинальная производительность аппаратных средств процессора в основном зависит от двух параметров – такта (или частоты) синхронизации и среднего количества тактов, необходимых для выполнения команды. Частота синхронизации определяется технологией аппаратных средств, а среднее количество тактов на команду определяется функциональной организацией процессора и архитектурой системы команд. Комплексная производительность характеризуется ещё одним параметром – количеством выполняемых команд в единицу времени, которое определяется архитектурой системы команд, количеством и типом команд, принадлежащих исполняемой прикладной программе, и технологией компиляторов. Для измерения времени работы

процессора на данной программе используется специальный параметр – время процессора (центрального процессора) – *CPU time*.

Одной из довольно часто встречаемых единиц измерения НП процессора является *MIPS* – миллион команд в секунду. Однако, использование *MIPS* в качестве показателя для сравнения различных вычислительных систем наталкивается на три проблемы. Первая – *MIPS* зависит от набора команд процессора, что затрудняет сравнение ВС с процессорами, имеющими разные системы команд. Вторая – *MIPS* даже одного и того же процессора меняется от одной прикладной программы к другой. Третья – *MIPS* может меняться по отношению к производительности в противоположную сторону в зависимости от функциональной организации процессора. Например, у процессора, имеющего в своём составе сопроцессор с плавающей точкой, в среднем на каждую команду требуется большее количество тактов синхронизации, чем у процессора только с целочисленной арифметикой. Поэтому последние выполняют большее количество команд в единицу времени и, следовательно, имеют более высокий рейтинг *MIPS*, хотя общее время выполнения прикладной задачи значительно больше, чем у первых. Подобное явление наблюдается и при использовании оптимизирующих компиляторов, когда в результате оптимизации сокращается количество выполняемых в программе команд, рейтинг *MIPS* уменьшается, а производительность увеличивается.

При решении научно-технических задач, в которых интенсивно используется арифметика с плавающей точкой, производительность процессора оценивается в *MFLOPS* - миллион элементарных арифметических операций над числами с плавающей точкой в секунду. Как единица измерения *MFLOPS* предназначена для оценки производительности только операций с плавающей точкой, и поэтому не применима вне этой ограниченной области. По мнению многих программистов, одна и та же программа, исполняемая на разных машинах, будет реализовываться через разное количество команд, но число операций с плавающей точкой будет неизменным. Однако следует обратить внимание на следующее. Во-первых, наборы операций с плавающей точкой не совместимы на различных процессорах. Во-вторых, рейтинг *MFLOPS* меняется на смеси быстрых и медленных операций с плавающей точкой. Например, программа со 100% операций сложения будет иметь более высокий рейтинг, чем программа со 100% операций деления. Решение обеих проблем заключается в том, чтобы взять «каноническое» или «нормализованное» число операций с плавающей точкой из исходного текста программы и затем поделить его на время выполнения. Например, авторы тестового пакета «Ливерморские циклы» [*http://www.citforum.ru/hardware/svk/glava3.shtml*] вычисляют для программы количество нормализованных операций с плавающей точкой в соответствии со следующими соотношениями (см. табл.1.8.1).

Таблица 7.1. Количество нормализованных операций с ПТ тестового пакета "Ливерморские циклы"

Реальные операции с плавающей точкой	**Нормализованные операции с плавающей точкой**
Сложение, вычитание, сравнение, умножение	1
Деление, квадратный корень	4
Экспонента, синус,...	5

Таким образом, рейтинг реальных *MFLOPS* отличается от рейтинга нормализованнных *MFLOPS*, который часто приводится в литературе по суперкомпьютерам.

Теперь вернёмся к системе тестов *SPEC*. Корпорация *SPEC* выполняет две основные функции:

1. Разрабатывает тестовые пакеты.
2. Собирает и публикует официальные результаты тестов.

Разработчики тестов отказались от использования стандартных абсолютных единиц типа *MFLOPS* или *MIPS*. Вместо этого используются собственные относительные единицы *SPEC*. Результаты «нормализуются» по отношению к аналогичным результатам на так называемой «эталонной» машине. Это рабочая станция *Sun Ultra* 5/10 (процессор *UltraSPARC* 11 с тактовой частотой 300*MHz*). На данной машине прогон всех тестов *CPU* 2000 (смотри ниже) занимает примерно двое суток.

Также *SPEC* предлагает и такие основные продукты:

- *CPU* 2000 - тесты вычислительной производительности (ранее использовался пакет тестов *CPU*95);
- *JVM*98 - виртуальный тест *Java*-машины;
- *HPC*96 - тесты для высокопроизводительных систем: приложение сейсмической обработки *SPECseis*96 (*Seismic*), приложение вычислительной химии *SPECchem*96 (*GAMESS*) и приложение моделирования климата *SPECclimate* (*MM*5).

CPU 2000 - это тестовый пакет, разработанный подразделением *Open Systems Group* (*OPG*) корпорации *SPEC* для оценки производительности микропроцессоров и вычислительных систем. Этот пакет состоит из двух групп тестов – *CINT* 2000 для оценки производительности на целочисленных операциях и *CFP* 2000 для оценки производительности на операциях с плавающей точкой. Буква «*C*» в названиях тестов означает, что тесты являются компонентными, в отличие от тестов производительности системы в целом. Обе группы тестов ориентированы на оценку работы

микропроцессоров, подсистемы кэш-памяти и оперативной памяти, а также компиляторов. В пакет *CINT* 2000 входят следующие тесты: 11 тестовых приложений, написанных на языке *C*, и один тест (252*eon*) на *C++* (см. табл.1.8.2).

Таблица 7.2. Набор тестов пакета *CINT* 2000

Название	**Краткое описание задачи**
164.gzip	Утилита сжатия данных (*gzip*)
174.vpr	Приложение для расчёта *FPGA*-кристаллов
176.gcc	Компилятор *C*
181.mcf	Приложение для решения задачи потока минимальной стоимости в сети
186.crafty	Программа для игры в шахматы
197.parser	Синтаксический разбор для естественного языка
252.eon	Трассировка лучей
253.perlbmk	Интерпретатор языка *Perl*
254.gap	Вычислительная задача из теории групп
255.vortex	Объектно-ориентированная база данных
256.bzip2	Утилита сжатия данных (*bzip2*)
300.twolf	Задача позиционирования и маршрутизации

В набор *CFP* 2000 входят 14 тестовых приложений, из которых 6 написано на языке *Fortran* 77, 4 на языке *Fortran* 90 и 4 на *C++* (см. табл.1.8.3). Более подробное описание этих задач доступны на веб-сайте *SPEC*.

Результаты *CPU* 2000 представляются в четырёх вариантах (см. табл.1.8.4).

Таблица 7.3 Набор тестов пакета *CFP* 2000

Название	**Краткое описание задачи**
168.wupwise	Задача квантовой хромодинамики
171.swim	Гидродинамическая задача моделирования для "мелкой" воды
172.mgrid	Многосеточная решалка для трёхмерного потенциального поля
173.applu	Решение параболических/элептических дифференциальных уравнений
177.mesa	Трёхмерная графическая библиотека (*Mesa3D*)
178.galgel	Гидродинамическая задача: анализ колебательной нестабильности
179.art	Моделирование нейронной сети
183.equake	Моделирование землетрясения методом конечных элементов
187.facerec	Задача распознавания лиц на графических изображениях
188.ammp	Задача вычислительной химии
189.lucas	Задача теории чисел (проверка простоты)
191.fma3d	Моделирование crush-тестов методом конечных элементов
200.sixtract	Моделирование ускорителя элементарных частиц

01.apsi	Моделирование ускорителя элементарных частиц

Таблица 7.4. Метрики тестового пакета *CPU* 2000

Метрика	**Максимальная оптимизация**	**Стандартная оптимизация**
Время выполнения одной итерации теста	*SPECxx* 2000	*SPECxx base* 2000
Количество итераций за фиксированное время	*SPECxx rate* 2000	*SPECxx base rate* 2000

Здесь *xx* – это или *int*, или *fp*, «*base*» – это метрика, которая соответствует компиляции тестовых программ с некоторыми «базовыми» опциями оптимизации, одинаковыми для всех тестов в тестовом пакете и такими же, что используются и при компиляции пользовательских программ. Метрики, в которых отсутствует «*base*», являются необязательными и соответствуют наилучшей оптимизации, которую производитель может обеспечить для каждого конкретного теста. Метрика типа «*rate*» позволяет оценить суммарный объём вычислений, который компьютер может выполнить за определённое время. То есть, например, на *SMP* – системе позволяется запустить несколько копий одного теста и в качестве результата выдать суммарное количество итераций, выполненное всеми процессорами за определённое фиксированное время. Метрики, в которых отсутствует «*rate*», оценивают просто «скорость» вычислений.

Показатели SPEC вычисляются следующим образом:

Например, показатель *SPECint* 2000 определяется так:

SPECint 2000 = RefTime / MeasuredTime, где

RefTime – время исполнения теста на эталонной машине, *MeasuredTime* – время исполнения на тестируемой машине.

Таким образом, смысл этого показателя – в относительном ускорении по сравнению с эталонной машиной.

Показатель *SPECint rate* 2000 вычисляется следующим образом:

$SPECint\ rate\ 2000 = N \times (RefTime / MeasuredTime) \times (60 \times 60 \times 24 / RefJobTime)$,

где N – число запущенных копий (итераций) теста, 60×60×24 – это число секунд в сутках, а величина *RefJobTime* принята равной 9600.

В качестве примера можно привести показатели *SPEC* для однопроцессорных систем, являющихся лидерами на 25 мая 2000 года (см. табл. 1.8.5).

Таблица 7.5. Показатели *SPECint* однопроцессорных систем

Система	**Процессор**	***SPECint* 2000**	***SPECint base* 2000**
Compaq AlphaServer DS20	*Alpha 21264A/667MHz*	444	424
Compaq AlphaServer ES40	*Alpha 21264A/667MHz*	433	413

Intel VC820	*Pentium 3/1GHz*	410	407

Лидерами по показателю *SPECfp* 2000 являются следующие однопроцессорные системы (см. табл.1.8.6).

Таблица 7.6. Показатели *SPECint* однопроцессорных систем

Система	**Процессор**	***SPECfp* 2000**	***SPECfp base* 2000**
Compaq AlphaServer DS20E	*Alpha 21264A/667MHz*	577	514
Compaq AlphaServer ES40	*Alpha 21264A/667MHz*	562	500
Compaq AlphaServer DS20	*Alpha 21264/500MHz*	422	383

Системы на базе *Pentium 3/1GHz* по производительности на операциях с плавающей точкой занимают только 10-е место с показателями 284/273. Таким образом, лидером по тестам *CPU* 2000 является процессор *Alpha21264A/667MHz*.

Лидерами по показателям «*rate*» одновременно на целочисленных операциях и операциях с плавающей точкой являются следующие многопроцессорные системы (см. табл. 1.8.7).

Таблица 7.7. Показатели класса «*rate*» многопроцессорных систем

Система	Процессор	Число ЦП	*CFP* 2000 (*rate*)	*CINT* 2000 (*rate*)
HP 9000 Model N4000	*PA-8600/552MHz*	8	23.0	32.7
HP 9000 Model N4000	*PA-8600/552MHz*	4	14.4	17.0
SGI 2200	*R12000/400 MHz*	4	13.2	15.4

Лекция 7

8. Классификация вычислительных систем

Классификация ВС весьма сложная проблема и, к сожалению, до сих пор окончательно не решена из-за большого разнообразия ВС, причём это разнообразие постоянно растёт, что приводит к появлению новых признаков классификации. К настоящему времени существует несколько признаков классификации, которые применяются, в зависимости от назначения ВС, либо самостоятельно, либо в определённой их совокупности. Например, ВС часто разделяют в зависимости от их структурной организации на следующие классы [С6]:

1. массивно-параллельные системы (МПС; на английском – *MPP*);
2. симметричные мультипроцессорные системы (СМС – *SMP*);
3. системы с неоднородным доступом к памяти (СНДП – *NUMA*);
4. параллельные векторные системы (ПВС – *PVP*);
5. кластерные системы (КС).

   Подробнее эти системы будут рассмотрены позднее.

   Иногда ВС различают:

1. по числу одновременно обрабатываемых программ – однопрограммные и многопрограммные (мультипрограммные);
2. по режиму функционирования – системы реального времени (обычно применяются в системах управления технологическими процессами) и универсальные (не привязанные к реальному времени);
3. по возможности изменения структуры – пассивные (с жёсткой структурой, созданной в расчёте на определённый класс задач) и активные, способные настраиваться на вновь поступившую прикладную задачу;
4. по типу используемых процессоров – однородные и неоднородные;
5. по условиям использования – индивидуального пользования, использования в пакетном режиме и коллективного пользования и т.д.

Применяя известную или создавая новую классификацию, необходимо ответить на главный вопрос – для кого предназначается данная классификация и на решение какой задачи направлена. Так разделение систем обработки данных на персональные ЭВМ, рабочие станции, серверы, большие универсальные ЭВМ, мини супер-ЭВМ, супер-ЭВМ позволяет пользователю примерно прикинуть производительность и стоимость приобретаемого изделия для решения требуемого класса задач. Разработчику классификация должна подсказать возможные пути совершенствования СОД и в этом смысле она должна быть достаточно содержательной. А удачной классификацию можно назвать тогда, когда с её помощью могут быть определены направления поиска новых структур и архитектур средств вычислительной техники.

В [3] авторы приводят оригинальный подход к классификации СОД, который основан на выделении «основополагающих, неделимых признаков, имеющих генный характер». Определены пять генетических признаков классификации систем обработки данных. Этими признаками являются:

1. способ представления информации;
2. тип запоминающей среды;
3. способ доступа к информации;
4. совмещение функций хранения и обработки информации;
5. способ организации процесса обработки информации.

Перечень признаков и видов их реализации приводятся в таблице 10.1.

Таблица 8.1. Генетические признаки классификации СОД и виды их реализации

Генетический признак	Реализация признака
Способ представления информации	Цифровой (символьный); Образный
Тип запоминающей среды	Дискретная; Распределённая (непрерывная)
Способ доступа к информации	Адресный; Ассоциативный
Совмещение функций хранения и обработки информации	Раздельные; Совмещённые
Способ организации процесса обработки информации	Алгоритмический; Самоорганизующийся

Рассмотрим данные признаки более подробно. В своей работе [3] авторы отмечают, что известны два способа представления информации: символьное в виде абстрактных цепочек, цифр, букв или других символов; образное в виде чувственных образов. В силу целого ряда объективных причин в технических СОД информация представлена как последовательность символов двоичного алфавита. Такая форма представления информации в наибольшей степени соответствует возможностям современной технологии создания элементов, структурной и логической организации СОД. Однако с ростом сложности решаемых задач, их многомерности всё в большей степени проявляются недостатки одномерной символьно-отвлечённой формы представления информации как чужеродной нашему мышлению. Важно также отметить, что символьная форма представления информации сложилась как технология работы с готовой информацией, знаниями, базирующаяся на формальной логике. Далее авторы делают вывод о том, что в биологических системах, и, прежде всего, в памяти человека, информация представлена в более сложном виде. Известно, что человек использует два механизма мышления. Во-первых, так называемое символьное или алгебраическое мышление позволяет работать с абстрактными цепочками символов, с которыми связаны семантические и

прагматические представления. Другой механизм - образное, геометризованное мышление позволяет работать с чувственными образами и представлениями об этих образах. Образы обладают гораздо большей информативностью, чем символы.

С точки зрения такого признака классификации как «тип запоминающей среды», авторы выделяют базовое свойство, позволяющее разделить все существующие запоминающие среды на два класса. Это свойство определено как *способ фиксации информации* в запоминающей среде. В известных запоминающих средах информация фиксируется либо в локальной области (точке) среды, либо распределена по всей поверхности или всему объёму среды. Запоминающие среды, в которых в любой фиксированный момент времени можно выделить локальную область, точку среды, хранящую наименьшую единицу информации (бит) классифицируются как *дискретные запоминающие среды*. Запоминающие среды, в которых невозможно выделить локальную область, хранящую наименьшую единицу информации, и информация в которых распределена по всей поверхности или всему объёму среды классифицируются как *распределённые (непрерывные) запоминающие среды*. К ним относятся, например, нейронные сети и голографические среды.

Что касается признака «способ доступа к информации», то стоит отметить лишь следующее. Способ доступа, при котором информация в запоминающей среде находится путём указания её местоположения в среде, называется *адресным*. Способ доступа к информации в запоминающей среде, базирующийся на механизме ассоциации, получил название *ассоциативного* способа доступа.

Четвёртый признак классификации «совмещение функций хранения и обработки информации» разделяет СОД на два класса. Первый связан с разделением хранения и обработки информации, что приводит к созданию самостоятельных устройств, предназначенных для хранения и обработки информации. В настоящее время этот класс систем является доминирующим в области обработки данных. Второй класс использует принцип переноса процесса обработки непосредственно в память ВС. Это становится возможным только с применением ассоциативного способа доступа к информации, который позволяет совместить функции хранения и обработки данных в одном устройстве.

Пятый признак по существу выделяет системы с традиционным алгоритмическим, программным способом организации вычислительного процесса (процесса обработки информации) и системы, базирующиеся на самоорганизации процесса обработки информации за счёт её непрерывного самообучения. К этому классу систем можно отнести нейрокомпьютеры.

Широко распространённым признаком классификации параллельных систем, введённым М. Дж. Флинном (а по нашему определению любая ВС является параллельной), есть соотношение между числом потоков команд и числом потоков данных, имеющих место в вычислительной системе при

выполнении одной общей или нескольких независимых задач. Под потоком команд понимается последовательный ряд команд, выполняемых системой, а под потоком данных - последовательность данных, вызываемых потоком команд, включая и промежуточные результаты. Считается, что в ВС могут присутствовать одиночные (О) и множественные (М) потоки команд (К) и данных (Д). В этом случае существуют четыре класса систем, из которых три последних являются параллельными:

1. ОКОД (*SISD*) – одиночный поток команд, одиночный поток данных (*последовательная ВС*);
2. ОКМД (*SIMD*) – одиночный поток команд, множественный поток данных (*параллельная ВС*);
3. МКОД (*MISD*) – множествнный поток команд, одиночный поток данных (*параллельная ВС*);
4. МКМД (*MIMD*) – множественный поток команд, множественный поток данных (*параллельная ВС*).

Множественность потока команд предполагает одновременное выполнение нескольких команд, множественность потока данных – одновременную обработку нескольких данных. Следует отметить, что и та, и другая множественность может быть реализована, если в ВС присутствует множественность аппаратных средств (процессоры, модули памяти) и соответствующее программное обеспечение.

ОКМД. Различные данные обрабатываются по одной программе (одни и те же действия ОК – одиночный поток команд, с разными данными Д1,Д2,Д3,Д4 – множественный поток данных, см. рис. 8.1). Очевидно, что такая система, составленная из большого числа процессоров, может обеспечить очень высокую производительность только на тех задачах, при решении которых все процессоры могут делать одну и ту же работу.

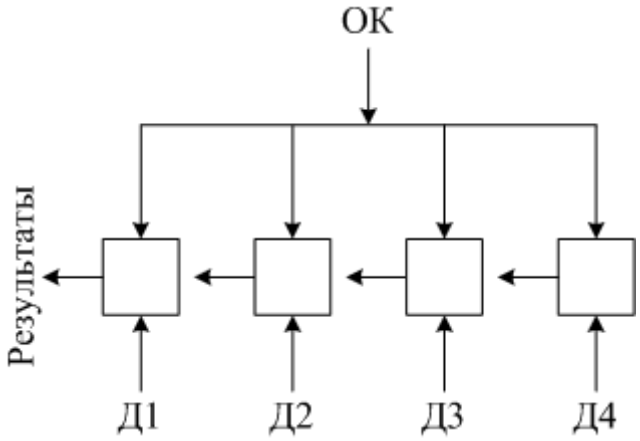


Рис.8.1 Обобщённая схема системы ОКМД

ВС (ОКМД) может быть построена:

1. на основе векторных процессоров;
2. на основе матричных процессоров;
3. на основе ассоциативных процессоров.

МКОД. Одни и те же данные ОД (см. рис) проходят несколько этапов обработки:

экспресс обработка → первичная обработка → вторичная обработка…

Каждый этап должен быть выполнен за определённое время.

Эти системы асинхронны, т.к. время решения различно, у каждого процессора свой поток команд К1,К2,К3,К4.

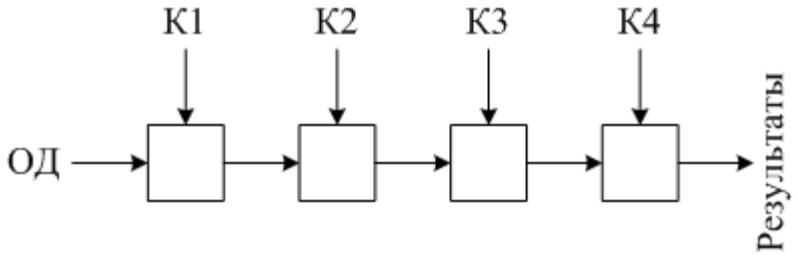


Рис.8.2 Обобщённая схема системы МКОД

Класс МКОД содержит один вид, который определён как магистральные (конвейерные) ВС, иногда их называют мультиконвейерными ВС.

Вышеуказанный признак классификации характеризует прежде всего организацию функционирования вычислительной системы и не раскрывает практические вопросы реализации взаимодействия компонент ВС при решении одной общей или нескольких независимых задач. Поэтому в последнее время появилось большое число расширений классификации Флинна. Одно из таких расширений подразумевает разбиение класса ОКМД на три вида. Это вычислительные системы с векторным потоком команд, ВС, построенные на основе матричных процессоров и ВС, построенные на основе ассоциативных процессоров, что представлено на рис. 8.3. Все эти системы будут рассмотрены при дальнейшем изложении материала.

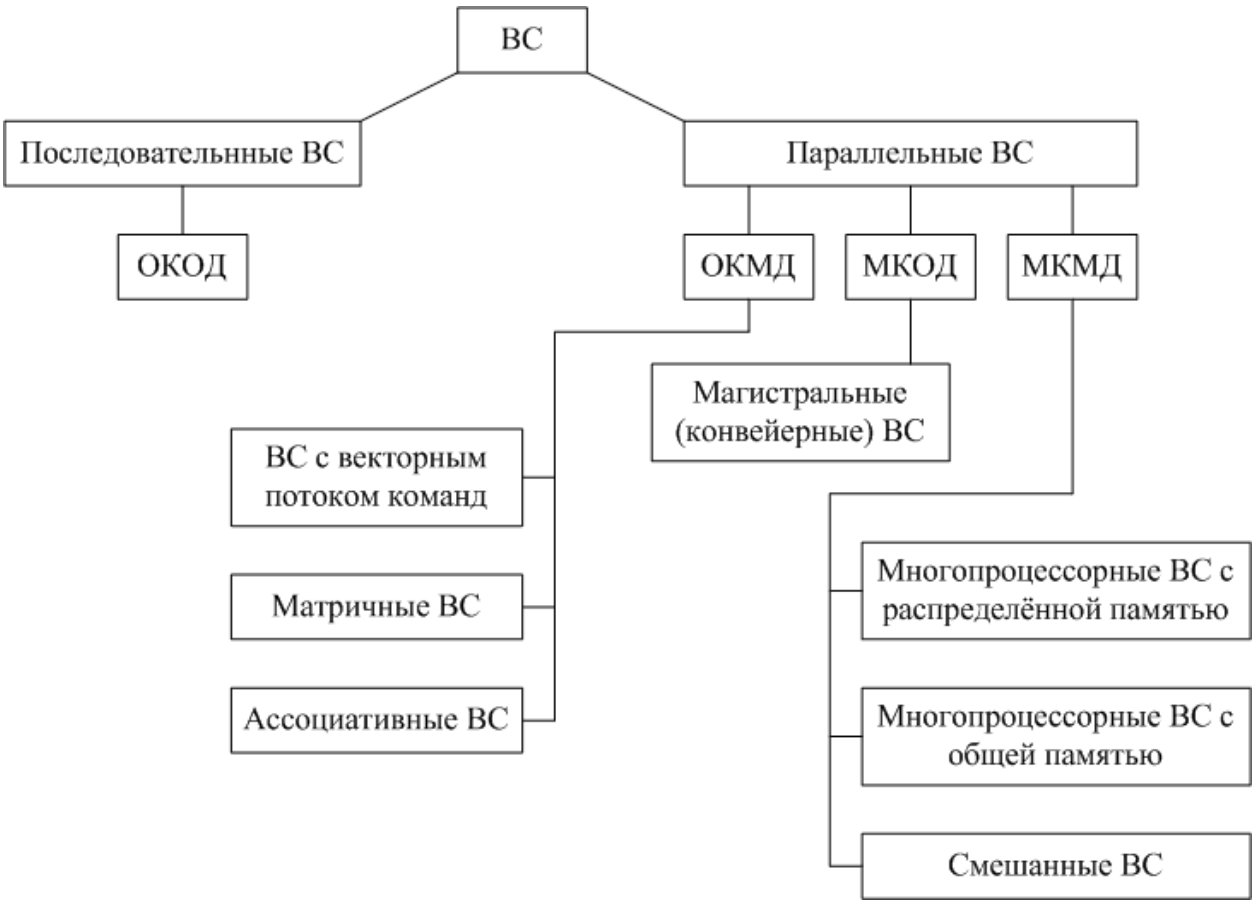


Рис. 8.3 Классификация вычислительных систем

Несмотря на то, что класс МКМД чрезвычайно широк, он разбивается на три вида – многопроцессорные ВС (МВС) с распределённой памятью, МВС с общей памятью и МВС со смешанной организацией памяти.

Большое число авторов предлагают различные классификации отдельных классов параллельных ВС. Например, Р. Хокни – известный английский специалист в области параллельных вычислительных систем, разработал свой подход к классификации систем, попадающих в класс МКМД, представленной на рис. 8.4. Основная идея классификации базируется на том, что множественный поток команд может быть обработан двумя способами: либо одним конвейерным устройством обработки, работающим в режиме разделения времени для отдельных потоков, либо каждый поток обрабатывается своим собственным устройством. Первая возможность используется в МКМД системах, которые автор называет конвейерными.

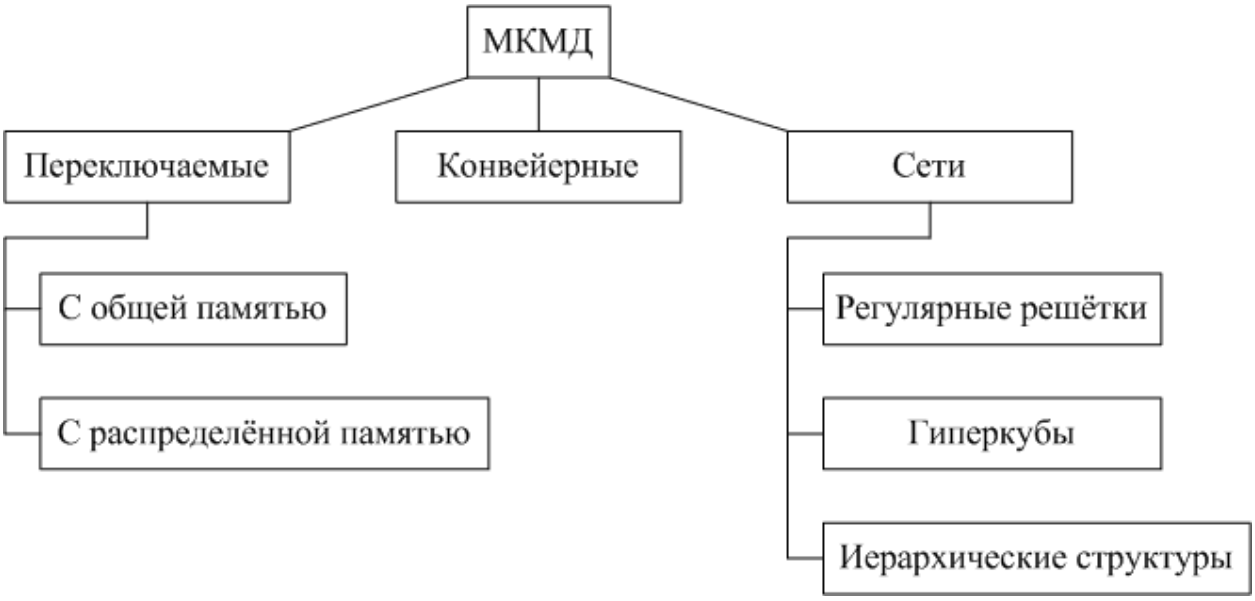


Рис. 8.4 Классификация Хокни

Архитектуры, использующие вторую возможность, в свою очередь опять делятся на два класса:

1. МКМД системы, в которых возможна прямая связь каждого процессора с каждым, реализуемая с помощью переключателя;
2. МКМД системы, в которых прямая связь каждого процессора возможна только с ближайшими соседями по сети, а взаимодействие удалённых процессоров поддерживается специальной системой маршрутизации через процессоры посредники.

Далее, среди МКМД систем с переключателем Хокни выделяет те, в которых вся память распределена среди процессоров как их локальная память. В это случае общение самих процессоров реализуется с помощью очень сложного переключателя, составляющего значительную часть системы. Такие системы носят название *систем с распределённой памятью*. Если память разделяемый ресурс, доступный всем процессорам через переключатель, то такие МКМД системы являются *системами с общей памятью*. В соответствии с типом переключателей можно проводить классификацию и далее: простой переключатель, многокаскадный переключатель, общая шина. Многие современные вычислительные системы имеют как общую разделяемую память, так и распределённую локальную память. Такие системы Хокни рассматривает как гибридные МКМД с переключателем.

При рассмотрении МКМД систем с сетевой структурой считается, что все они имеют распределённую память, а дальнейшая классификация проводится в соответствии с топологией сети.

Позднее Е. Джонсон предложил проводить классификацию МКМД систем на основе структуры памяти и реализации механизма взаимодействия и синхронизации между процессорами. По структуре оперативной памяти

существующие вычислительные системы делятся на две большие группы: либо это системы с общей памятью, прямо адресуемой всеми процессорами, либо это системы с распределённой памятью, каждая часть которой доступна только одному процессору. Одновременно с этим, и для межпроцессорного взаимодействия существует две альтернативы: через разделяемые переменные или с помощью механизма передачи сообщений. Исходя из таких предположений, можно получить четыре класса МКМД систем, уточняющих классификацию Флинна:

1. общая память – разделяемые переменные;
2. распределённая память – разделяемые переменные;
3. распределённая память – передача сообщений;
4. общая память – передача сообщений.

В 1988 году была сделана очередная попытка расширить классификацию Флинна и, тем самым, преодолеть её недостатки. Д. Скилликорн разработал подход, пригодный для описания свойств многопроцессорных систем и некоторых нетрадиционных архитектур, в частности, потоковых машин. Хотя автор излагает свою классификацию как классификацию архитектур вычислительных систем, на мой взгляд речь идёт о классификации структур ВС. Предлагается рассматривать архитектуру любой системы, как абстрактную архитектуру, состоящую из четырёх компонент:

1. процессор команд (*IP – Instruction Processor*) – функциональное устройство, работающее как интерпретатор команд; в системе, вообще говоря, может отсутствовать;
2. процессор данных (*DP – Data Processor*) – функциональное устройство, работающее как преобразователь данных, в соответствии с арифметическими операциями;
3. иерархия памяти (*IM – Instruction Memory, DM – Data Memory*) – запоминающее устройство, в котором хранятся команды и данные, пересылаемые между процессорами;
4. переключатель – абстрактное устройство, обеспечивающее связь между процессорами и памятью.

Функции процессора команд во многом схожи с функциями устройства управления последовательных машин и, согласно, Д. Скиллкорну, сводятся к следующему:

1. на основе своего состояния и полученной от *DP* информации *IP* определяет адрес команды, которая будет выполняться следующей;
2. осуществляет доступ к *IM* для выборки команды;
3. получает и декодирует выбранную команду;
4. сообщает *DP* команду, которую надо выполнить;
5. определяет адрес операндов и посылает их в *DP;*
6. получает от *DP* информацию о результате выполнения команды.

Функции процессора данных делают его, во многом, похожим на арифметическое устройство традиционных процессоров:

1. *DP* получает от *IP* команду, которую надо выполнить;
2. получает от *IP* адреса операндов;
3. выбирает операнды из *DM*;
4. выполняет команду;
5. запоминает результат в DM;
6. возвращает в *IP* информацию о состоянии после выполнения команды.

В терминах определённых таким образом основных частей системы можно представить структуру любой параллельной системы, при этом автор выделяет четыре типа переключателей, без какой-либо явной связи с типом устройств, которые они соединяют:

1-1 – переключатель такого вида связывает пару функциональных устройств;

n-*n* – переключатель связывает *i*-ое устройство из одного множества устройств с *i*-ым устройством из другого множества, т.е. фиксирует попарную связь;

1-*n* – переключатель соединяет одно выделенное устройство со всеми функциональными устройствами из некоторого набора;

n×*n* – каждое функциональное устройство одного множества может быть связано с любым устройством другого множества, и наоборот.

Примеров подобных переключателей можно привести очень много. Так все матричные процессоры имеют переключатель типа 1-*n* для связи единственного процессора команд со всеми процессорами данных. В машинах семейства *Connection Machine* каждый процессор данных имеет свою локальную память и, следовательно, связь будет описываться как *n*-*n*. В то же время, каждый процессор команд может связываться с любым другим процессором, поэтому данная связь будет описана как *n*×*n*.

Классификация Д. Скилликорна состоит из двух уровней. На первом уровне она проводится на основе восьми характеристик:

1. количество процессоров команд (IP);
2. число запоминающих устройств (модулей памяти) команд (IM);
3. тип переключателя между IP и IM;
4. количество процессоров данных (DP);
5. число запоминающих устройств (модулей памяти) данных (DM);
6. тип переключателя между DP и DM;
7. тип переключателя между IP и DP;
8. тип переключателя между DP и DP.

В терминах данных характеристик машину *Connection Machine* 2 можно описать так: (1, 1, 1-1, *n*, *n*, *n*-*n*, 1-*n*, *n*×*n*).

На втором уровне классификации Д. Скилликорн просто уточняет описание, сделанное на первом уровне, добавляя возможность конвейерной обработки в процессорах команд и данных. Автор сформулировал три цели, которым должна служить хорошо построенная классификация:

1. облегчать понимание того, что достигнуто на сегодняшней день в области архитектур вычислительных систем, и какие архитектуры имеют лучшие перспективы в будущем;
2. подсказывать новые пути организации архитектур - речь идёт о тех классах, которые в настоящее время по разным причинам пусты;
3. подсказывать, за счёт каких структурных особенностей достигается увеличение производительности различных вычислительных систем; с этой точки зрения, классификация может служить моделью для анализа производительности.

Теперь попытаемся привести свой подход к построению классификации архитектур вычислительных систем (см. рис.8.5). Во введении было определено понятие архитектуры ВС как совокупность трёх категорий - сущность информационных потоков (ИП), характер их взаимодействия и способ обработки данных. Будем считать эти категории основными признаками классификации. Однако для более детальной проработки классификации необходимо ввести дополнительные признаки, которые уточняют вид архитектуры ВС. Рассмотрим первый признак. На первое место, на мой взгляд, необходимо поставить способ обработки данных, так как он определяет тип реализации непосредственно операций над данными в исполнительных устройствах процессора, и разбиение на классы архитектур по этому признаку не зависит от значений следующих признаков. При анализе способа обработки данных можно выделить две составляющие:

1. координата (обработка может быть горизонтальной, вертикальной и многокоординатной);
2. параллелизм (последовательная, последовательно-параллельная и параллельная обработка элемента потока данных).

Второй признак – сущность информационных потоков – определяет совокупность видов информационных потоков, циркулирующих в системе обработки данных. Например, можно выделить четыре вида ИП – данные, команды, функции и задания. В настоящее время можно определить три класса архитектур в соответствии с данной составляющей признака: (команды – данные), (функции – данные) и (задания – данные).

После того как определены ИП в СОД, необходимо уточнить характер их прохождения в исполнительных устройствах СОД. Характер прохождения может быть организован либо в виде конвейера, либо без такового. Причём конвейер, как было отмечено выше, реализуется на уровне операций, команд,

процессов, соответствующих, например, отдельным ветвям программы, и различным сочетаниям этих конвейеров.

Важной характеристикой архитектуры ВС является реализация типового контроля элементов информационных потоков. Введение типов устанавливает взаимно однозначное соответствие между объектом и его изображением в информационной среде и, по существу, устраняет последствие семантического разрыва неймановской архитектуры. Задание типов данных и реализация типового контроля может осуществляться либо программно, либо программно-аппаратно, либо аппаратно. Как было отмечено в предыдущих разделах, ещё одним узким местом неймановской архитектуры является так называемый побочный эффект. Эффективность смягчения этого эффекта зависит от способа реализации конструкций языков высокого уровня. Таких способов существует три: программный, программно-аппаратный и аппаратный.

Из оставшихся пяти признаков следует обратить внимание на следующие два. Один из них основной – характер взаимодействия потоков, который по существу разделяет архитектуру ВС на два класса.

Первый использует в том или ином виде неймановский принцип обработки, атрибутом которого является счётчик команд.

Второй – принцип управления данными, атрибутом этого типа архитектуры является признак готовности данных. Этот основной признак можно уточнить так – очередной элемент ИП данного типа инициирует активизацию порции информационного потока другого вида. Например, в неймановской архитектуре очередная команда содержит информацию о местоположении в памяти элементов потока данных, над которыми и производятся определённые действия. Ещё один признак, который необходимо уточнить, это реализация параллелизма. На примере рис.8.5 видно, что, если рассматривать параллелизм на уровне команд, то он может быть реализован либо статически (распределение команд по исполнительным устройствам процессора осуществляется на уровне компиляции заранее), либо динамически – то же распределение осуществляется устройством управления в процессе обработки данных.

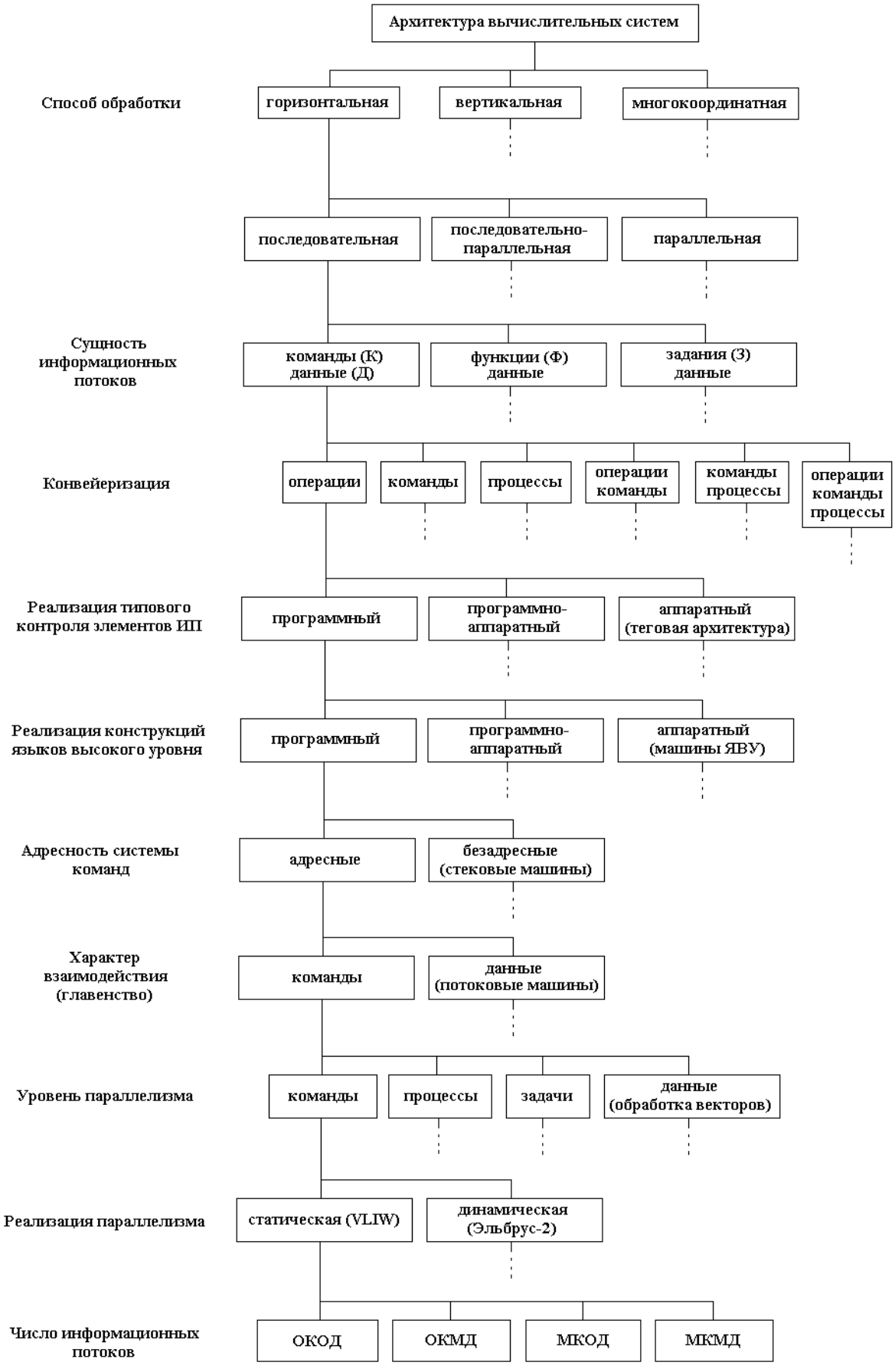


Рис. 8.5 Архитектура вычислительных систем

9. Режимы обработки данных ВС

Одним из важнейших путей повышения производительности вычислительных систем является использование того или иного режима функционирования в зависимости от класса решаемых задач и требований к процессу их выполнения. Существуют два режима работы ВС – однопрограммный и мультипрограммный (многопрограммный). По режиму обслуживания данные системы можно разделить на два вида – индивидуального и коллективного пользования. По способу приёма заданий на обработку различают пакетную обработку и обработку заданий поступающих независимо друг от друга в случайные моменты времени от разных пользователей. Пакетная обработка может производиться как в однопрограммном, так и многопрограммном режимах. Режим коллективного пользования – это форма обслуживания, при которой возможен одновременный доступ нескольких независимых пользователей к вычислительным ресурсам ВС. Системы коллективного пользования с квантованным обслуживанием называются системами с разделением времени [5].

9.1 Пакетный режим функционирования ВС

В этом режиме пользователи не имеют непосредственного доступа к ВС. Подготовленные ими программы передаются по каналам связи в систему, где они накапливаются и оформляются в виде единого пакета. В момент, когда система готова принять на обработку пакет заданий начинается процесс обработки по заранее составленному расписанию.

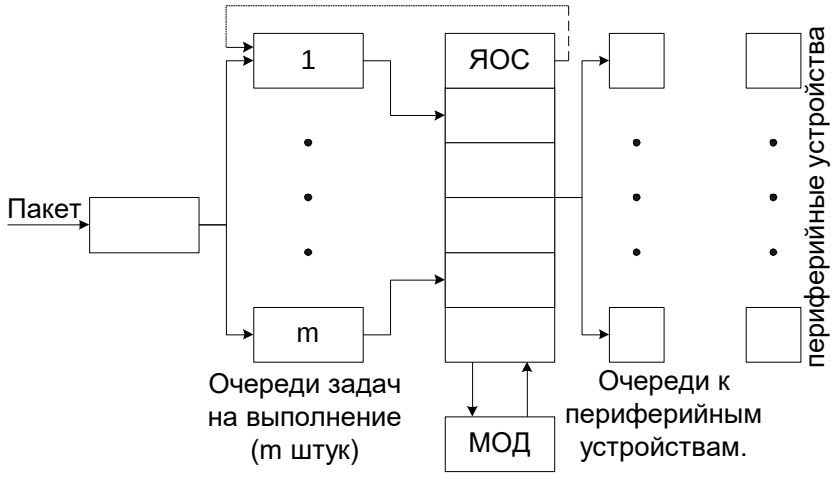


Рис.9.1 Схема пакетной обработки задач

Каждый пакет имеет паспорт, в котором содержится число задач, их тип. В свою очередь каждая задача имеет свой паспорт, в котором указывается приоритет и др. необходимая информация.

МОД – модуль обработки данных.

ЯОС – ядро операционной системы.

Ядро ОС содержит планировщик, который определяет, какую задачу выполнять (с учётом приоритетов).

Если режим – однопрограммный, то пакеты выполняются последовательно.

В мультипрограммном режиме формируются очереди (на основе приоритетов задач). Ядро ОС выделяет определённую область памяти для каждой задачи. После того, как задача выполнилась, формируются очереди на пользование периферийными устройствами.

В пакетном режиме применяется механизм *квантования*. Т.е. каждой задаче выделяется время исполнения (время процессора) Δ и если за это время задача не выполнится, то она поступает в конец очереди.

Пакетный режим используется очень часто, несмотря на высокие «накладные расходы».

Особенности пакетного режима:

1. Развитая система прерываний (необходимо прерывать выполнение задачи и сохранять её вектор, перейти к другой задаче, а затем снова возвращаться к выполнению этой задачи). Это приводит к значительным «накладным расходам».
2. Необходима защита памяти от доступа к данным задачи из других задач.
3. Необходимы специальные привилегированные команды (команды защиты), доступные только системе.
4. Требуется специальный датчик времени (таймер) для реализации механизма квантования.

Мультипрограммная пакетная обработка обеспечивает высокую степень загрузки ресурсов ВС, но при этом из-за отсутствия непосредственной связи между системой и пользователем производительность и эффективность труда самих пользователей снижается по сравнению с индивидуальным обслуживанием.

9.2 Режим разделения времени (потоковый режим)

В общем случае на вход ВС поступает случайный поток задач от нескольких источников (пользователей).

Вычислительная система с разделением времени (СРВ), работающая в мультипрограммном режиме, в определённой последовательности обслуживает пользователей, выделяя программе (заданию) каждого пользователя некоторый интервал времени (квант обслуживания). Если в течение выделенного программе кванта времени её обработка не заканчивается, программа прерывается и становится в очередь программ, ожидающих обработку.

Порядок распределения между пользователями основного ресурса СРВ – времени процессоров – устанавливается дисциплиной квантованного обслуживания. Выбор значения кванта времени определяется двумя

основными факторами – уменьшение времени отклика на запросы пользователя и увеличение загрузки основного ресурса ВС. Малое значение кванта времени, выделяемое программе ведёт к снижению эффективности использования технических средств системы, так как переключение программы из пассивной фазы в активную сопровождается служебными операциями, доля которых при этом в общем времени работы системы возрастает. Значение кванта времени выбирается в результате разумного компромисса, с тем, чтобы время отклика на запрос, и потери производительности от переключения активности программ были приемлемыми.

Возможно два варианта дисциплин обслуживания:

а) одноочередная дисциплина обслуживания, упрощенная схема СРВ с которой представлена на рис. 9.2. Вновь поступающие от пользователя программы ставятся в конец очереди.

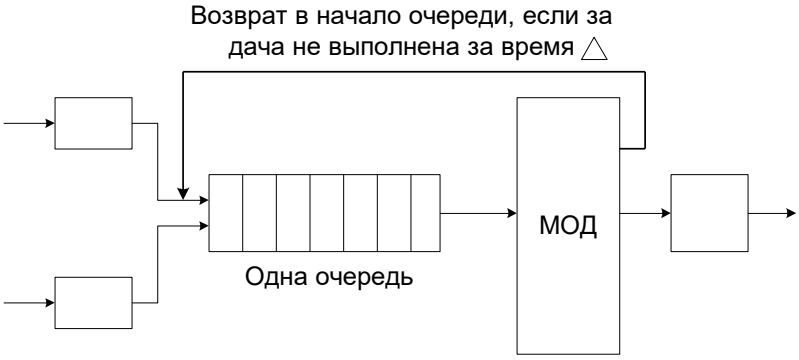


Рис.9.2 Схема системы с режимом разделения времени с одноочередной дисциплиной обслуживания

Здесь обязательно используется режим квантования. Если задача не решена за время Δ, то она отправляется в конец очереди.

Время Δ выбирается исходя из параметров ВС.

Недостатком такой дисциплины обслуживания является очень длительное время решения больших задач.

б) многоочередная дисциплина обслуживания, упрощенная схема СРВ с которой представлена на рис. 9.3.

Для сокращения потерь времени, связанных с переключением программ, и обеспечения приоритета для коротких программ используют дисциплину обслуживания с несколькими очередями

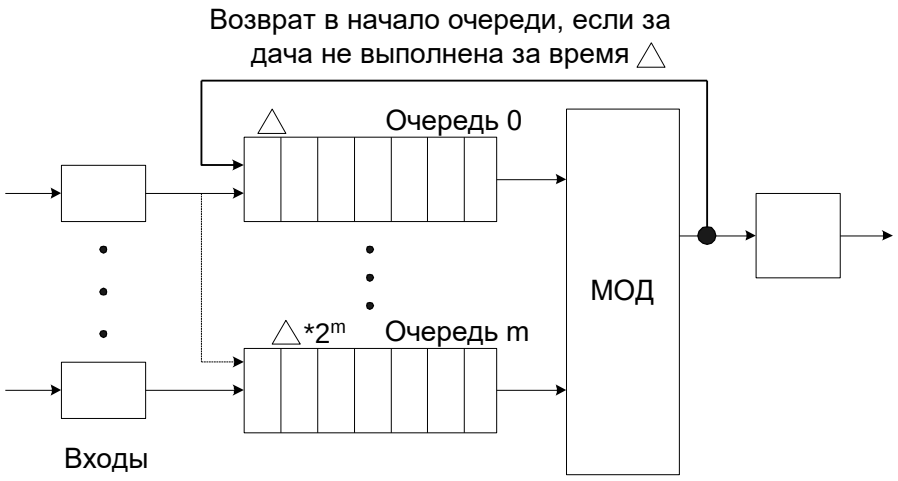


Рис. 9.3 Схема системы с многоочередной дисциплиной обслуживания

У каждой очереди свой приоритет (приоритет соответствует номеру очереди: 0- наибольший приоритет, m– наименьший приоритет).

У задачи может быть задан приоритет, либо его устанавливает ОС, основываясь на данных, полученных из паспорта задачи (время выполнения задачи и требуемые ресурсы). Если время выполнения задачи не указано, то оно оценивается пропорционально длине кода программы. Чем дольше исполняется программа, тем ниже приоритет, но Δ в таких очередях больше.

По приоритету задача направляется в соответствующую очередь.

Если задача имеет наивысший приоритет, но не успевает выполниться, в отведенный квант времени (Δ), то вступает в силу механизм динамических приоритетов: *если задача не решена за время Δ, то она «проваливается» в очередь с более низким приоритетом*. Т.е. пользователь устанавливает максимальный приоритет, а ОС его постепенно понижает.

Номер очереди, куда следует поместить задачу, определяется по следующей формуле:

$$m = \log_2\left(\left[\frac{l_p}{l_\Delta}\right] + 1\right)$$

m- номер очереди, куда следует поместить задачу,

l_p - длина кода программы в байтах,

l_Δ -длина кода программы в байтах, которая может быть передана из оперативной памяти во внешнюю или обратно за квант времени Δ.

9.3 ВС реального времени

В ряде применений вычислительных систем, таких, как контроль и управление различными технологическими процессами, на обработку вводимых данных и выдачу результатов накладываются жёсткие ограничения на время выполнения задачи, диктуемые темпом развития процессов в контролируемом объекте.

Вычислительной системой реального времени (системой, работающей в масштабе реального времени) называется система, соединенная с некоторым внешним объектом и обрабатывающая поступающую в неё информацию о его состоянии достаточно быстро, для того, чтобы результат обработки мог использоваться для воздействия на протекание процессов в объекте.

На рис 9.4 представлена упрощённая структурная схема автоматизированной системы управления с модулем обработки данных МОД. Принцип обработки в таких системах носит итерационный (периодический) характер, данные D поступают на вход системы через равные промежутки времени и именно в течение этого промежутка времени должны быть выработаны управляющие воздействия Y на объект.

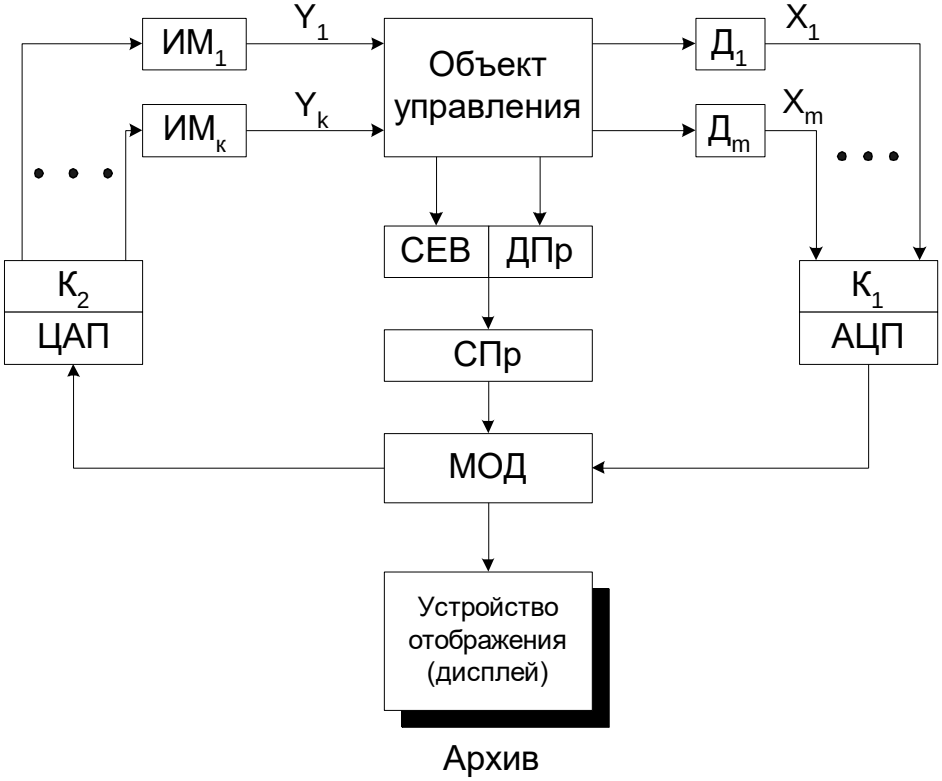


Рис.9.4 Схема системы реального времени

$ИМ_1 \ldots ИМ_К$- исполнительные механизмы.
$Д_1 \ldots Д_К$ – датчики.
СЕВ – система единого времени.

ДПр – датчик прерываний.
СПр – система прерываний.
МОД – модуль обработки данных.
К1,К2 –коммутаторы
АЦП – аналого-цифровой преобразователь
ЦАП – цифро-аналоговый преобразователь

Таким образом, главной особенностью вычислительных систем реального времени является способность принять данные, как правило, с большого числа датчиков, преобразовать их в цифровую форму, привязать к реальному отсчёту времени (с помощью системы единого времени), обработать, выдать управляющие воздействия, архивировать данные и постоянно отображать состояние объекта на устройстве отображения за фиксированный интервал времени. Второй главной особенностью таких систем является требование высокой надежности.

Лекция 8

10. Системы класса MIMD (МКМД)

Системы данного класса делятся, как это было отмечено выше на два основных вида – системы с общей и распределенной памятью. Обычно в качестве синонима систем первого вида используется термин «многопроцессорные вычислительные системы», второго вида «многомашинные вычислительные системы» [С6]. В МКМД системах каждый их процессор выполняет свою программу (или часть большой программы) достаточно независимо от других процессоров. Большой интерес к системам данного класса определяется двумя факторами:

1. свойства систем МКМД дают большую гибкость – при наличии адекватной поддержки со стороны аппаратных средств и программного обеспечения МКМД может работать как однопользовательская система, обеспечивая высокопроизводительную обработку данных для одной прикладной задачи, как многопрограммная система, выполняющая множество задач параллельно, и как комбинация этих возможностей;
2. система МКМД может использовать все преимущества современной микропроцессорной технологии на основе строгого учета соотношения производительность/стоимость. В действительности практически все современные МКМД системы строятся на тех же процессорах, что и персональный компьютер, рабочие станции и сервера.

Одной из отличительных особенностей МКМД систем является сеть обмена, с помощью которой процессоры соединяются друг с другом или с памятью. Модель обмена для таких систем настолько важна, что многие характеристики производительности и другие оценки выражаются отношением времени обработки к времени обмена, соответствующим решаемым задачам. Существуют две основные модели межпроцессорного обмена : одна основана на передаче сообщений, другая на использование общей памяти. В многопроцессорной системе с общей памятью один процессор осуществляет запись в конкретную ячейку, а другой производит считывание из этой ячейки памяти. Чтобы обеспечить согласованность данных и синхронизацию процессов, обмен часто реализуется по принципу взаимно исключающего доступа к общей памяти методом «почтового ящика». В многомашинной системе процессоры получают доступ к совместно используемым данным посредством передачи сообщений по сети обмена. Эффективность схем коммуникаций зависит от протоколов обмена, характеристик основных сетей обмена (в том числе каналов связи), пропускной способности памяти.

Таким образом, базовой моделью вычислений в МКМД системах является совокупность независимых процессов, эпизодически обращающихся к разделяемым данным. Существует достаточно большое количество вариантов этой модели. На одном конце спектра – это модель распределенных вычислений, в которой программа делится на довольно

большое число параллельных задач, состоящих из множества подпрограмм. На другом конце спектра – модель потоковых вычислений, в которой каждая операция в программе может рассматриваться как отдельный процесс. Такая операция ждет своих входных данных (операндов), которые должны быть переданы ей другими процессами. По их получении операция выполняется, и полученное значение передаётся тем процессам, которые в нём нуждается. На рис. 10.1 представлена упрощённая схема МКМД системы.

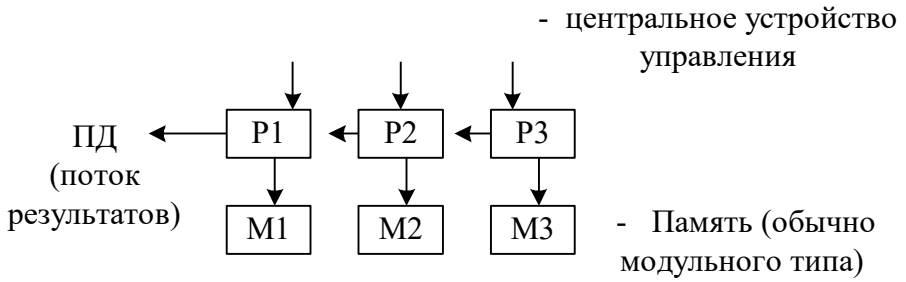


Рис. 10.1 Упрощённая схема системы МКМД

В реализации любой модели вычислений кто-то должен управлять всем этим процессом. Это м.б. специальный процессор (тогда он должен быть мощнее, чем каждый из P 1,2,3, так как он должен решать сложную задачу назначения). Это система с *централизованным управлением.* Ее минус в высокой стоимости.

Роль управляющего процессора могут играть несколько процессоров (В одной из систем Cray, например, входят 8 обрабатывающих процессоров и 4 управляющих, т.е. 2x1). В ОКМД всегда одно ЦУУ, но здесь будет несколько таких устройств. Один из управляющих процессоров назначается главным – обеспечивая взаимодействие 4-х управляющих процессоров.

Минусы:

- Сложная подготовка задач к параллельным вычислениям, ведь «простая» программа здесь не будет эффективна.
- Сложное и дорогое ЦУУ

Свойства MIMD:

- Иерархия управления (управляющий процессор, в каждом процессоре свое управление)
- Несколько уровней памяти (cashe 1,2,3; RAM; HD)
- Иерархия коммутации (магистрали между процессорами и магистрали между узлами)

Как было отмечено выше МКМД делятся на:

- Системы с общей памятью (UMA)
- Системы с распределенной памятью

- Системы со «смешанной» памятью (NUMA - Non Uniform Memory Access).

В системах со «смешанной» памятью модули памяти физически располагаются в разных процессорах, но адресное пространство у них общее, поэтому доступ к своей локальной памяти быстрее, чем к чужой).

11. ВС с общей памятью (UMA)

К данной группе систем МКМД относятся системы с общей (разделяемой) основной памятью, объединяющие до нескольких десятков (обычно не более 32) процессоров. Сравнительно небольшое число процессоров в таких системах определяется наличием одной шины, соединяющей эти процессоры. Таким образом, при наличии у процессоров кэш-памяти достаточной ёмкости высокопроизводительная шина и общая память могут удовлетворить обращения к ней поступающие от нескольких процессоров. Поскольку имеется единственная общая память с одним и тем же временем доступа для любого процессора, поэтому такие системы и получили название UMA (Uniform Memory Access). Довольно часто, и не всегда обосновано, при анализе систем с общей памятью (в том числе и систем на основе векторных процессоров) затраты на обмен не учитываются, так как проблемы обмена в значительной степени скрыты от программиста. Однако накладные расходы на обмен в этих системах имеются и определяются конфликтами шин, памяти и процессоров. Чем больше процессоров добавляются в систему, тем больше процессов соперничают при использовании одних и тех же ресурсов, что приводит к большему числу конфликтов. Модель системы с общей памятью очень удобна для программирования, так как мало отличается от программирования для однопроцессорной системы [4]. В случае неудовлетворительных результатов применения системы с одной шиной, число их может быть увеличено. На рис. 1.11 представлена схема такой системы.

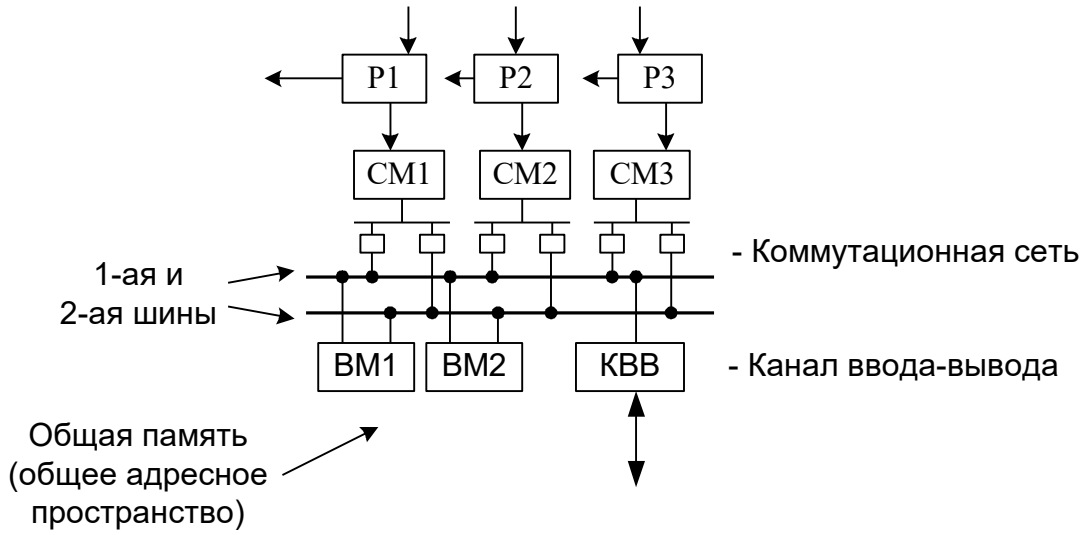


Рис. 11.1 Упрощённая схема системы МКМД с общей памятью

В состав данной системы входит арбитр (на каждую шину по одному арбитру – на рисунке не показаны), который регулирует доступ процессора к шине, т.к. процессор может одновременно работать только с одной шиной (т.к. кэш-память СМ у процессора - одна). Такая организация дает положительный эффект при наличии нескольких модулей памяти –ВМ.

Любой процессор в данной системе может быть управляющим.

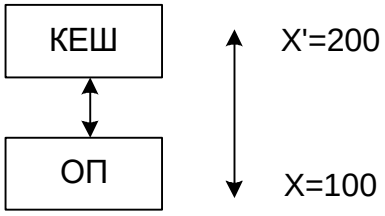
- Он анализирует паспорта задач (в каждом есть таблица связности и временные ограничения)
- Решает задачу назначения (в статическом режиме), т.е. выбирает стратегию назначения (либо из паспорта, либо с помощью своей экспертной системы).
- Ведёт мониторинг параллельных вычислений.

Преимущества систем с общей памятью следующие.

1. Совместимость с хорошо понятными используемые как в однопроцессорных, так и многопроцессорных системах, механизмами, которые используют для обмена общую память.
2. Простота программирования, когда модели обмена между процессами сложные или динамически меняются во время выполнения. Подобные преимущества упрощают процесс создания компиляторов.
3. Более низкая задержка обмена и лучшее использование полосы пропускания при обмене малыми порциями данных.
4. Возможность использования аппаратно управляемого кэширования для снижения частоты доступа к общей памяти. Без предполагаемой поддержки со стороны аппаратуры все обращения к разделяемой памяти потребуют привлечения операционной системы как для преобразования адресов так и защиты памяти.

12. Проблема когерентности в ВС с общей памятью

В системах с общей памятью кэши могут содержать как разделяемые («общие») , так и частные данные. Частные данные – это данные, которые используются одним процессором, в то время как разделяемые данные используются многими процессорами, по существу являясь причиной обменов между ними. Когда кэшируется элемент частных данных, их значение переносится в кэш для сокращения среднего времени доступа. Поскольку никакой другой процессор не использует эти данные, этот процесс идентичен процессу для однопроцессорной машины с кэш-памятью. Если кэшируются разделяемые данные, то их значение реплицируется и может содержаться в нескольких кэшах. Поэтому кэширование данных вызывает новую проблему – когерентность кэш-памяти.



Одна и та же переменная имеет разные значения в кэш-памяти и оперативной памяти. Это и есть нарушение когерентности.

Если процессор один, то рано или поздно он

все поправит, но если несколько процессоров и одна оперативная память, хранящая данные одной большой задачи, то налицо нарушение когерентности, причем сразу у нескольких процессоров, которым нужно сообщить об этом нарушении. Поэтому, на стадии программирования необходимо помнить об этом и предусмотреть способы смягчения этой проблемы. Существует два механизма решения этой проблемы:

1. *Наблюдение* – каждый процессор наблюдает за движением данных, помеченных признаком достоверности, тогда процессор обратится за данными в ОП, а не в свою кэш-память. Причем, наблюдение ведется постоянно. Этот механизм позволяет не использовать ложные данные. Механизм наблюдения за магистралью заимствован из сетей.
2. *Справочник* – считается единым справочником, но фактически распределен (находится у каждой СМ). В справочнике хранится информация обо всех данных. В случае недостоверности общих данных в справочнике выставляется, что это данное недостоверно и передается сообщение об этом справочникам других процессоров. Отличие этого механизма состоит в том, что мы не отлавливаем информацию, а получаем

Существуют следующие механизмы замены:

1. *Обнуление* (во всех IBM). Если общие данные изменились в i-м процессоре, то i-й процессор инициирует (сообщает) другим процессорам, что у них этого блока нет (в любой СМ есть АЗУ, в котором хранятся номера блоков, которые есть в СМ), т.е. обнуляем часть таблицы адресов (с данными не работаем), поэтому процессор обратится к СМ и увидит, что требуемого блока нет и обратится за ним к ОП. Признак недостоверности сохраняется в ОП до тех пор пока данные не будут обновлены (процесс обновления достаточно долгий).
2. *Обновление.* Если данные изменены, то необходимо посмотреть в справочнике и передать всем «свежие» данные. Это быстрее, но сложнее с точки зрения аппаратуры.

Лекция 9

13. ВС с распределенной памятью

В данном разделе рассматриваются так называемые крупномасштабные системы с распределенной памятью, т.е. каждый процессор имеет свою локальную память (ЛП)– для того чтобы поддерживать большое количество процессоров приходится распределять основную память между ними, в противном случае полосы пропускания памяти просто может не хватить для удовлетворения запросов, поступающих от очень большого числа процессоров. На рис. 13.1 показана структура такой системы. Для простоты на рис. 13.1 в качестве коммуникационной сети (КС) изображена общая шина.

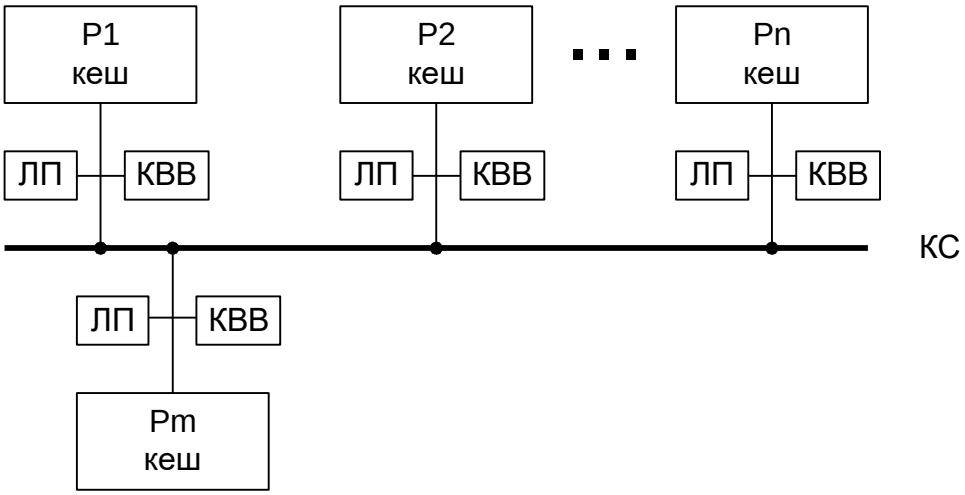


Рис. 13.1 Упрощённая схема системы МКМД с распределённой памятью

В таких системах в рамках одной локальной памяти может находится несколько процессоров, тогда такое образование называется вычислительным узлом системы. Распределение памяти между отдельными узлами системы имеет два главных преимущества. Во-первых, это эффективный с точки зрения стоимости способ увеличения полосы пропускания памяти, поскольку большинство обращений могут выполняться параллельно к локальной памяти в каждом узле. Во-вторых, это уменьшает задержку обращения (время доступа) к локальной памяти. Обычно устройства ввода/вывода, так же как и память, распределяются по узлам. На рис. 13.1 КВВ – контроллер ввода/вывода. Таким образом, каждый вычислительный узел представляет собой отдельный компьютер, поэтому такие системы и называются многомашинными. Поскольку физически адресное пространство распределено по вычислительным узлам доступ к «чужой» памяти осуществляется через механизм передачи сообщений.

Основные преимущества обмена с помощью передачи сообщений являются.

1. Аппаратура может быть более простой, по сравнению с моделью разделяемой памяти.
2. Модели обмена понятны программистам.
   Однако, основные трудности возникают при работе с сообщениями, которые могут быть неправильно выровнены и сообщениями произвольной длины в системе памяти, которая обычно ориентирована на передачу выровненных блоков данных, организованных как блоки кэш-памяти.

При оценке любого механизма обмена критичными являются три характеристики производительности систем обмена:

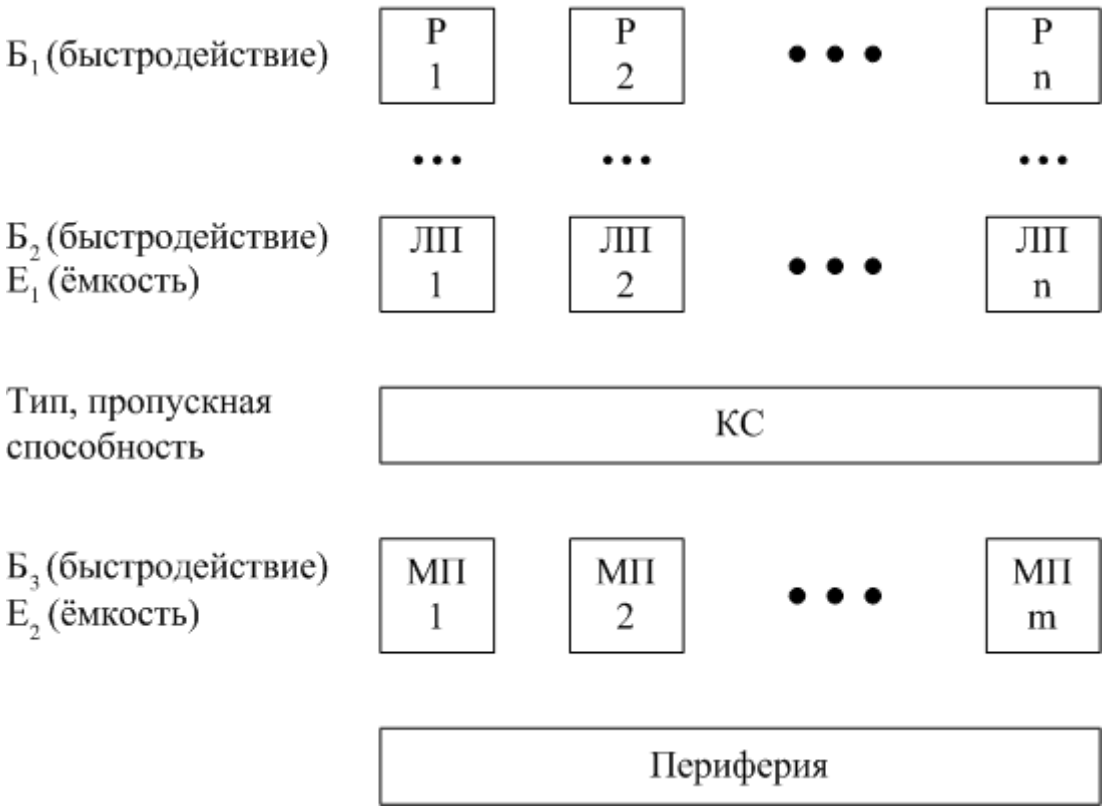
1. Полоса пропускания – в идеале полоса пропускания механизма обмена будет ограничена полосами пропускания процессора, памяти и системы межсоединений.
2. Задержка – в идеале задержка должна быть настолько мала, насколько это возможно. Для её определения критичны накладные расходы аппаратуры и программного обеспечения, связанные с инициированием и завершением обмена.
3. Упрятывание задержки – насколько хорошо механизм скрывает задержку путем перекрытия обмена с вычислениями или другими обменами.

Лекция10

14. Постановка задачи назначения

Объект исследования – это ВС, которая в одном случае организована как система с общей памятью, а в другом как система с распределенной памятью, и прикладные задачи, которые поступают для решения на вход ВС. Для каждого случая задача назначения должна решаться с учетом характеристик выбранной системы и параметров задач. Предмет исследования – это способ размещения работ (например, вычислений, передач), необходимых при выполнении, поступивших на вход ВС прикладных задач, на выделенные ресурсы ВС. Выбор способа размещения работ получил название «решение задачи назначения». Этой проблеме посвящены лабораторные работы, которым посвящены методические указания.

На рис. 14.1 представлены ресурсы ВС, участвующие в выполнении прикладных задач.



P – процессор
ЛП – локальная память процессора
КС – коммутационная сеть
МП – модуль оперативной памяти

Рис. 14.1 Ресурсы вычислительной системы

Множество процессоров – решающее поле. Каждый процессор может обладать своей локальной памятью LM, являющейся элементом единой памяти системы, тогда система определяется как система с распределенной памятью. Либо все процессоры имеют общую память, организованную как автономная оперативная многомодульная память ВМ. Связь в такой системе между ее компонентами и внешним миром осуществляется с помощью коммутационной сети.

Считается, что ВС однородная, т.е. все процессоры и модули памяти одинаковые, причём $Б_1=Б_2$, т.е. локальная память не вносит задержку в работу процессора. Это важное допущение!

Оперативная память в ВС строится по модульному принципу, причём число модулей *m* не превышает число шин (при шинной структуре КС). В ВС с общей памятью (ОП) имеет место единое адресное пространство, обращение к которому осуществляется посредством выполнения команд записи считывания. ОП – это общий ресурс, разделяемый процессорами. В ВС с распределённой памятью, в которой имеет место распределённое адресное пространство, обмен данными осуществляется с помощью передачи сообщений, поэтому такие ВС иногда называют *ВС с сетевой архитектурой.*

Прикладные задачи – внешняя среда по отношению к ВС.

Способ поступления прикладных задач на вход ВС:

1. *набор прикладных задач* – на вход поступает от 1 до *L* задач, оформленных в виде пакета;
2. *случайный поток задач* – на вход поступает одна или несколько задач в случайные моменты времени.

14.1 Модели представления задач

Наиболее распространена ярусно-параллельная форма (ЯПФ) представления задач. В этом случае прикладная задача представляется в виде направленного графа, в котором каждая вершина отображает некоторую часть (сегмент) задачи, а дуги – информационные связи между ними.

Число ярусов определяет высоту графа, а максимальное число вершин яруса – ширину. Ширина графа влияет на выбор количества процессоров, необходимых для решения данной задачи. Каждая вершина графа взвешивается числом операций, необходимых для выполнения соответствующего сегмента задачи, а дуги объемом данных, передаваемых по соответствующей информационной связи. Это *внешние параметры* задачи инвариантные относительно вычислительной системы, на которой она будет выполняться.

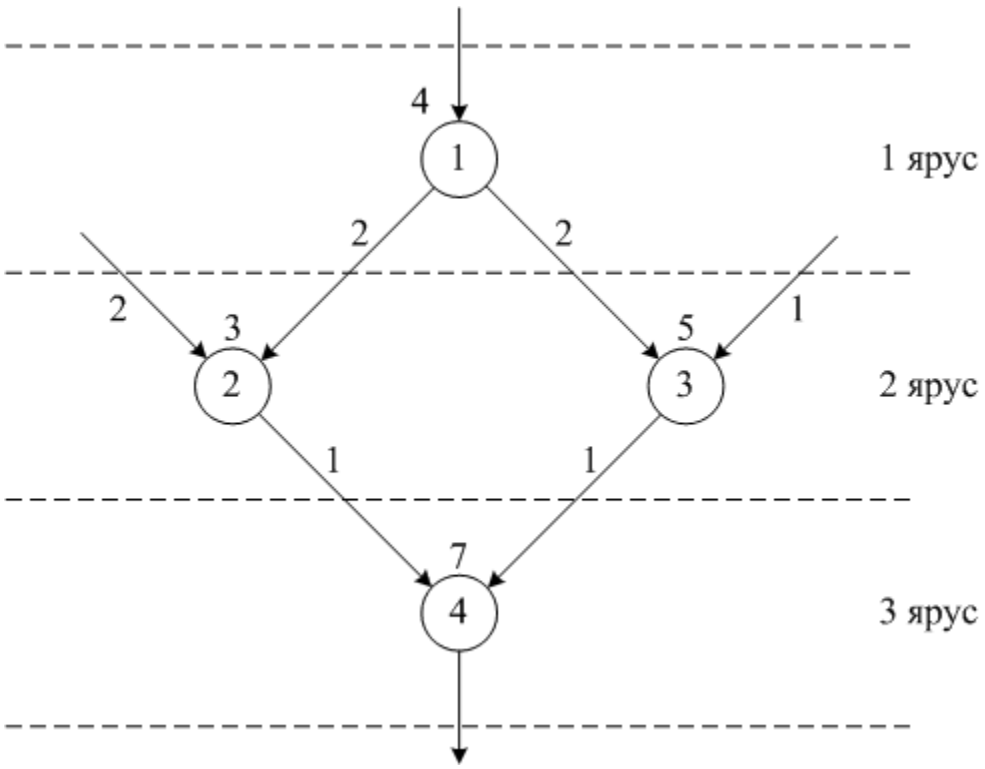


Рис. 14.2 Пример ярусно-параллельной формы

Внутренние параметры – это время выполнения сегмента задачи, соответствующего вершине графа, на данном типе процессора – $t_{обр.\ узла}$ и время передачи данных $t_{передачи}$ по каналу связи с заданной пропускной способностью, соответствующего дуге графа. На рис. 14.2 изображен граф задачи, состоящий из 4 вершин, номер вершины внутри кружка, сверху кружка цифра, соответствующая времени выполнения данного сегмента задачи в условных единицах, цифра у дуги – время передачи данных в тех же условных единицах.

По соотношению времени выполнения сегмента задачи и времени передачи данных задачи можно разделить на:

1. слабосвязные ($t_{обр.\ узла} >> t_{передачи}$);
2. среднесвязные ($t_{обр.\ узла} \approx t_{передачи}$);
3. сильносвязные ($t_{обр.\ узла} << t_{передачи}$);

Постановка задачи назначения в общем виде рассматривается как одна из двух задач оптимизации :

1. при заданных ресурсах ВС (число процессоров, число каналов передачи данных) распределить части прикладных задач по этим ресурсам (рассматривая заданный тип ВС (с общей или распределённой памятью) и определенный набор прикладных задач), так, чтобы время решения набора задач ($T_{решения}$)→min;
2. при заданном (допустимом) времени решения набора задач ($T_{доп}$) распределить части прикладных задач по ресурсам ВС (рассматривая заданный тип ВС (с общей или распределённой памятью) и

определенный набор прикладных задач) так, чтобы найти минимальные ресурсы, на которых возможно решение задач за допустимое время.

Производные показатели:

1. $K_{уск}$ – *коэффициент ускорения решения задачи*

   $K_{уск} = \frac{T_{1P}}{T_{nP}}$, где T_{1P} – время решения задачи на одном процессоре,

   T_{nP} – на n процессорах.
2. $K_з$ – *коэффициент загрузки оборудования*

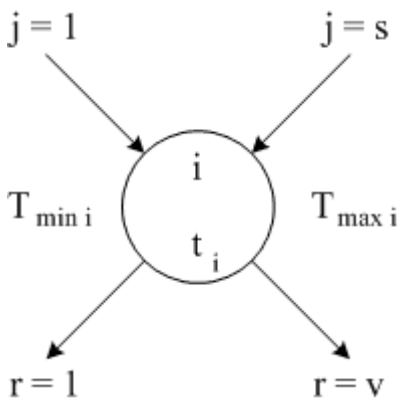
   $K_з = \frac{T_{обработки}}{T_{решения}}$, $T_{обработки}$ –время, в течении которого данное оборудование занято выполнением функций при решении задачи (набора задач), $T_{решения}$ – общее время решения задачи (набора задач).

14.2 Алгоритм поиска критического пути графа

Для оценки минимально возможного времени решения задачи применяется *алгоритм поиска критического пути*, суть которого заключается в определении минимально возможного и максимально возможного времени начала выполнения узлов графа.

При начальном проходе от начальной вершины графа к конечной определяется минимально возможное время начала выполнения каждого узла по формуле:

$$T_{min\ i} = \max_{j=1} (T_{min\ j} + t_j + T_{ji}) \qquad (1)$$



s – число вершин-предшественников i-ой вершины;

$T_{min\ i\ (j)}$ – минимально возможное время начала выполнения i-ой (j-ой) вершины;

t_j – время выполнения j-ой вершины;

T_{ji} – время передачи данных между вершинами j и i, которому приписывается одно из значений множества $\{0, \tau_{ji}, 2\tau_{ji}\}$ в зависимости от способа организации памяти, где τ_{ji} – время передачи данных между вершинами j и i, задаваемое на исходном графе задачи.

При повторном анализе графа при проходе от конечной вершины к начальной определяется максимально возможное время начала выполнения вершины по формуле:

$$T_{min\ i} = \min_{r=1} (T_{max\ r} - t_i - T_{ir}) \qquad (2)$$

v – число вершин-последователей i-ой вершины;

t_i – максимально возможное начало выполнения i-ой (r-ой) вершины;

T_{ir} – время передачи данных i-ой вершины вершине r, значение которому присваивается аналогично T_{ji}.

При этом для начальной вершины (вершин) $T_{min\ i} = 0$, а для конечной вершины минимально возможное время начала её выполнения совпадает с максимально возможным временем начала выполнения – $T_{max\ i} = T_{min\ i}$

Вершина является *критической*, если выполняется равенство: $T_{max\ i} = T_{min\ i}$

Критическим путём графа является множество последовательных вершин, начинающихся входной вершиной и заканчивающихся выходной вершиной, у которых $T_{max\ i} = T_{min\ i}$, в котором для каждой пары вершин графа соотношения (1) и (2) определяются соседней вершиной в паре. Т.е. это такой путь, который имеет максимальное время выполнения среди всех путей графа задачи. И, следовательно, он определяет минимально возможное время выполнения всей задачи при неограниченных ресурсах вычислительной системы, на которой она выполняется.

Пример. Определение критического пути.

Допустим, каждой вершине графа рис. 14.3 поставлено в соответствие её время выполнения на данном типе процессора в машинных тактах, указанное над вершиной. Тогда, по указанному выше алгоритму, определены минимально (слева от вершины) и максимально (справа от вершины) времена начала соответствующей вершины.

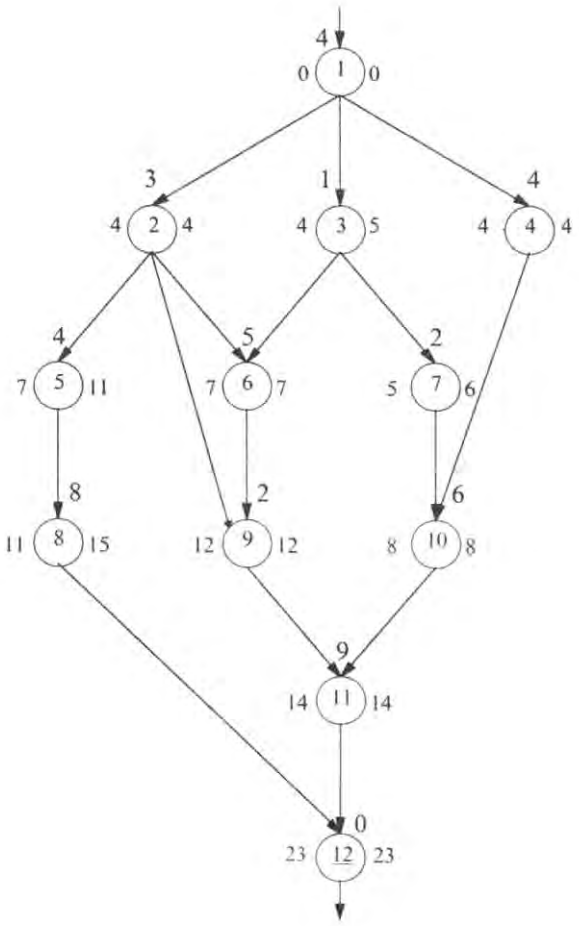


Рис. 14.3 Граф задачи

Критический путь по определению будет: 1-2-6-9-11-12

T_{min} = 23 мт (машинных тактов) - минимально возможное время решения задачи.

$T_{max} = \sum_{j=1}^{N} T_j$ = 48 (N – число вершин графа, т.е. T_{max} – время решения задачи на одном процессоре).

Допустим, что необходимо решить задачу за $T_{доп}$ =25мт. Для того, чтобы не делать полного перебора при выборе числа процессоров n, определим начальное его значение, необходимое для решения задачи за $T_{доп}$ ($T_{доп} \geq T_{min}$, иначе задачу невозможно решить за требуемое время):

$n = T_{max} / T_{доп} = 48 / 25 \approx 2$

14.3 Применение стратегий назначения при решении задачи назначения

Как назначить готовые к исполнению вершины графа задачи на процессоры?

Например, это можно сделать на основе *теории расписания*, часто прибегая к *эвристическим методам*, позволяющим найти результат, близкий к оптимальному. Эти методы определяются следующими стратегиями выбора готовых к исполнению вершин и назначения их на свободные процессоры:

1. равновероятный выбор;
2. с максимальным временем исполнения узла;
3. с минимальным временем исполнения узла;
4. на основе принадлежности узла критическому пути.

Для автоматического решения задачи назначения граф задачи представляется в виде таблицы связности размерностью NxN, в ячейки которой заносятся признаки связи соответствующих вершин.

Будем считать, что на вход многопроцессорной вычислительной системы поступают задачи, относящиеся к классу слабосвязных. Тогда временем передачи данных можно пренебречь.

Рассмотрим задачу назначения на основе принадлежности критическому пути, а в случае наличия нескольких готовых к исполнению вершин, не принадлежащих критическому пути, стратегии с минимальным временем выполнения вершины для графа задачи, представленного на рис. На рис. 14.4 представлена диаграмма Ганта, на которой изображены временные диаграммы работы двух процессоров.

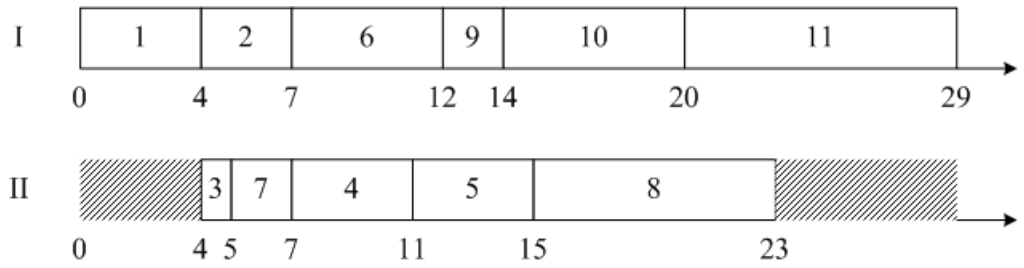


Рис. 14.4 Диаграмма Ганта

В данном случае не уложились в $T_{доп}$ =25 мт, следовательно, необходим третий процессор. Изменение стратегии назначения, скорее всего, не поможет, т.к. один процессор занят полностью, а другой вынужден какое-то время простаивать. Эмпирически доказано, что чем больше процессоров задействовано в решении задачи, тем больше простаивает каждый из них. Для рассматриваемого случая $T_{решения}$=29 мт, следовательно, $K_{уск}$ = 48/29, $K_з$ первого процессора равен 1 , а второго 19/29.

Лекция 11

14.4 Обеспечение надежности вычислений

Время простоя процессоров можно использовать для контроля исправности оборудования за счет дублирования выполнения одних и тех же вершин на разных процессорах (во время их простоя).

Для нашего примера $P_{обн.\ ош.} = 10/29$ – это максимально возможное значение вероятности обнаружения ошибки при полном заполнении времени простоя второго процессора (10 мт) решением узлов, выполняемых первым процессором.

Ошибка – событие, заключающееся в получении результата вычислений, отличного от правильного из-за сбоев и отказов оборудования.

Сбой – самоустраняющийся отказ, приводящий к кратковременному нарушению работоспособности.

Отказ – событие, заключающееся в постоянном нарушении работоспособного состояния объекта.

Пример заполнения времени ожидания (простоя) процессоров для графа задачи, изображенного на рис. 14.5 копиями уже выполненных вершин (времена выполнения вершин приняты одинаковыми для простоты), представлен на этом же рисунке.

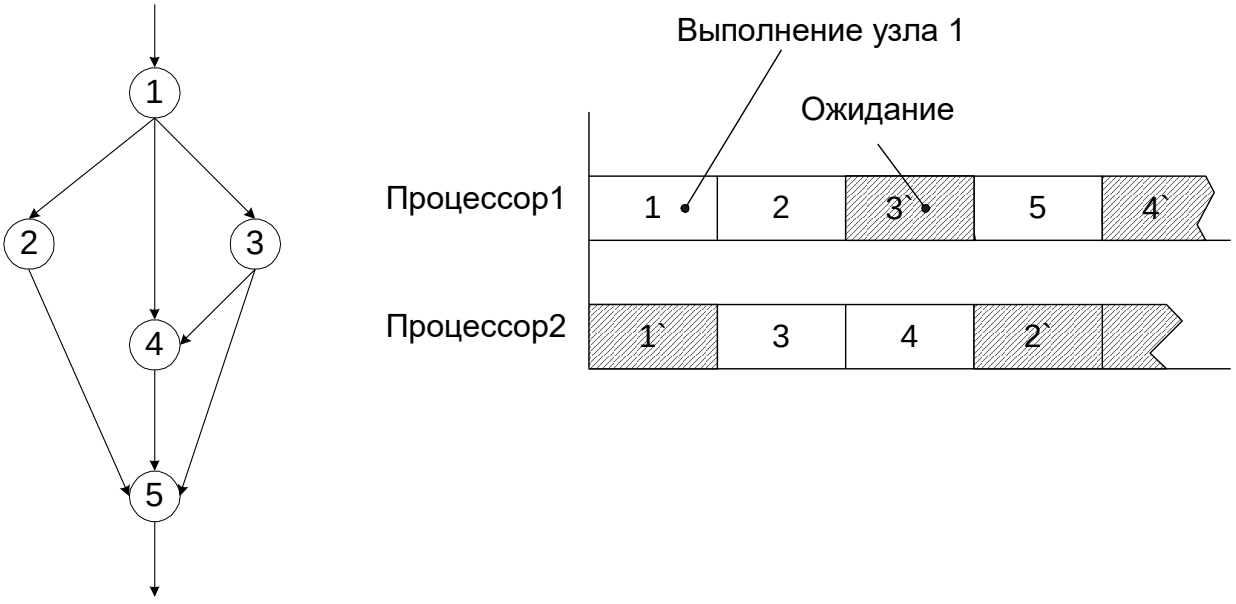


Рис. 14.5 Пример заполнения времени простоя

Если нужно обнаруживать ошибки с вероятностью $P_{обн.\ ош.}=1$, то требуется продублировать выполнением все вершины (дублируемые вершны должны выполняться на разных процессорах).

Обнаруживать ошибки можно:
А) с точностью до вершины, тогда граф задачи преобразуется так, как это показано на рис….., на котором показаны только 2 вершины из 5 соответствующие операции сравнения результатов;
Б) с точностью до всей задачи, тогда граф задачи преобразуется так, как это показано на рис.14.6 , при этом необходимо сравнение результатов для всей задачи.

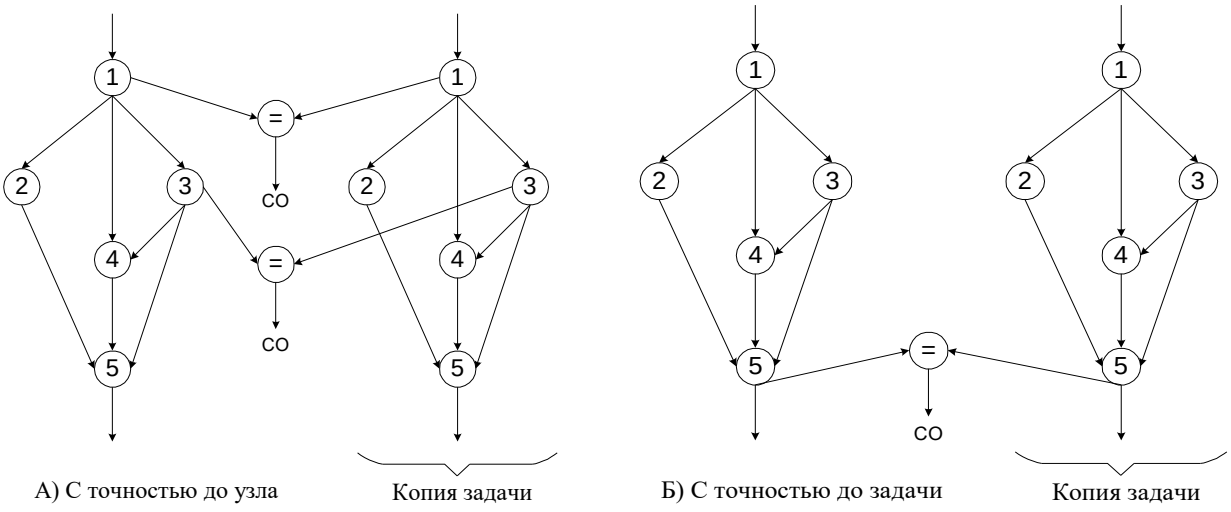


Рис. 14.6 Пример дублирования графа задачи

В варианте А матрица связности 15 на 15.
В варианте Б матрица связности 11 на 11.
СО – сигнал ошибки.

Помимо обнаружения ошибки, за счёт ещё большего увеличения количества используемого оборудования, а возможно и увеличения времени решения задачи можно сделать и исправление ошибки. В этом случае задача представляется в виде графа рис.14.7.

Таблица истинности МО			
X1	X2	X3	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

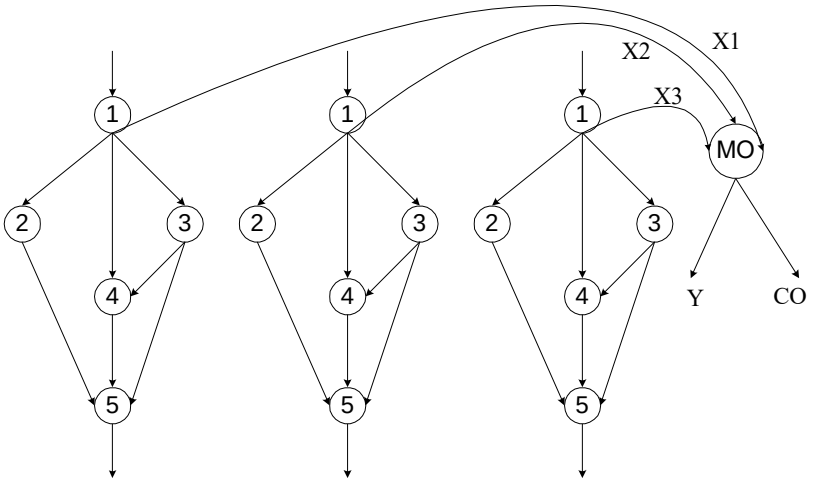


Рис. 14.7 Пример троирования графа задачи

СО- сигнал ошибки
МО- мажоритарный орган.
Y- результат (уже исправленный).

Однако, следует помнить, что отказать может и исправляющая система, поэтому вычисления ***не*** *могут быть абсолютно надёжными*.

14.5 Многозадачный режим функционирования Многопроцессорной Вычислительной Системы

Допустим на вход МВС поступил набор (пакет) задач рис. 14.8 , который необходимо выполнить.

Если эти задачи будут решаться последовательно (друг за другом), то тогда время решения набора задач равно сумме времён решения каждой задачи по отдельности.

$T_{рнз}=T_{рз1}+T_{рз2}$

Но при решении первой задачи могут появиться простои процессоров, которые можно заполнить решением второй задачи. Именно этим занимается так называемый планировщик многозадачного режима.

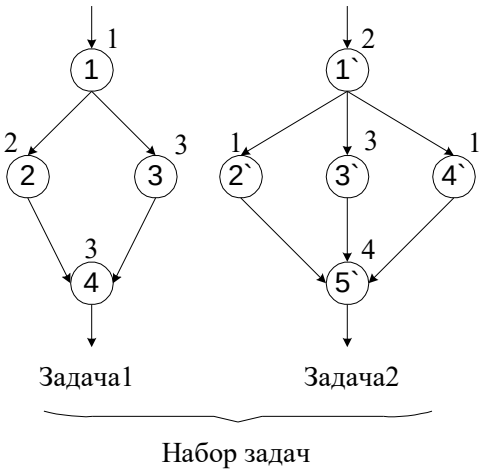


Рис. 14.8 Набор из двух графов задач

Задачи из набора могут быть как с равным приоритетом, так и с различным. Последовательность выбора вершины, готовой к исполнению, при выполнении набора задач определяется не только выбранной стратегией назначения, но и соответствующим критерием:

- $T_{зад.наб.}$ – заданное время решения набора задач.
- $T_{зад.\ пр.\ задач}$- заданный приоритет задач.

$$\left.\begin{cases} t_{\min 1} \to КП1 \\ t_{\min 2} \to КП2 \end{cases}\right\} \max \to T_{\min\, н} \qquad T_{\max\, н.з.} = \sum_{i=1}^{4} t_i + \sum_{j}^{5} t_j$$

т.е. минимальное время выполнения набора задач равно длине наибольшего критического пути, а максимальное время выполнения набора задач равно сумме времени выполнения всех вершин всех задач в наборе.

При поступлении в МВС, каждый набор задач снабжается паспортом. Паспорт набора задач содержит следующие данные:

- количество задач;
- приоритет задач;
- количество необходимых ресурсов;
- стратегия назначения.

Каждая задача также имеет свой паспорт, в котором присутствует таблица связности, необходимые ресурсы и др.

При решении задачи назначения, в зависимости от числа задач в наборе и их приоритетов, формируются очереди готовых к исполнению вершин

задач. В нашем случае получается 2 очереди выполнения вершин (с учётом приоритетов). Сначала рассматривается задача с высшим приоритетом.

Существует два механизма:

1. Готовая исполнению вершина выбирает себе процессор из очереди свободных.
2. Освободившийся процессор выбирает себе следующую вершину из очереди готовых.

Временная диаграмма примера (стратегия- максимальное время выполнения; задача 1 имеет высший приоритет) представлена на рис .

Рис. 14.9 Диаграмма Ганта при многозадачном режиме

$К_{заг.мзр\ i} \geq К_{заг.озр\ i}$

Коэффициент загрузки в многозадачном режиме всегда больше или равен коэффициенту загрузки в однозадачном режиме.

При выборе стратегии назначения для набора задач выбирается та, которая на конкретной задаче даёт наибольший выигрыш.

Например, анализируем стратегии минимального и максимального времени выполнения:

-для каждой задачи определяется время по min и max стратегиям и из большего вычитается меньшее (определяется выигрыш).

-для каждой задачи сравниваются эти выигрыши.

Есть два режима распределения вершин (составления расписания):

- *Статический*- распределение вершин осуществляется перед выполнением задачи.
- *Динамический*- распределение вершин осуществляется по ходу выполнения задачи.

Лекция 12

15. Системы коммутации

Существует большое число разнообразных способов соединений компонентов вычислительных систем [1]. Тип организации таких соединений зависит не только от требовании к их пропускной способности, но и от таких количественных характеристик ВС как число процессоров, модулей памяти, каналов ввода вывода, а также и от особенностей структурной организации самих ВС. Например, существенным является то каким компонентам необходимо связываться друг с другом (процессор-процессор, процессор-память). Так для систем класса ОКМД основной задачей параллельных вычислений на множестве процессоров является параллельная доставка данных (разных операндов) из общей памяти в локальную память каждого процессора. Чтобы связь между множеством процессоров и памятью была гибкой и достаточно быстродействующей, необходимо иметь большое число параллельно функционирующих линий связи со сложной системой их коммутации (СК). Как правило, для такого рода ВС прибегают к созданию специализированных СК на основе так называемых многоступенчатых коммутаторов, соответствующих их конкретному назначению. С одной стороны это приводит к значительному удорожанию ВС, а с другой стороны к обеспечению высокой пропускной способности СК.

Для систем класса МКМД важным вопросом при построении СК является то, как организована память таких ВС, т.е. это организация с общей памятью или с распределённой памятью. Так для систем с общей памятью, обладающей физически единым адресным пространством, и являющейся разделяемым ресурсом ВС, СК строится, как правило, с использованием одной или нескольких шин и соответствующих коммутаторов, встроенных в процессоры и модули памяти, а сам обмен данными осуществляется с помощью выполнения команд «записи» «считывания».Для систем с распределённой памятью характерно то, что обмен данными осуществляется по схеме «обмен сообщениями» и, следовательно, существует огромное число видов построения СК, начиная от применения общей шины, одноступенчатых и многоступенчатых коммутаторов и заканчивая построением сложных сетей коммутации, таких, как например, гиперкуб большой размерности.

Компоненты системы коммутации бывают 2-х типов:

1) активные (контроллеры шин, адаптеры, коммутаторы)
2) пассивные (магистрали, шины)

Таким образом, рассматривая СК и абстрагируясь от типа ВС, можно сделать вывод о том, что основными компонентами СК являются активные компоненты. (Коммутаторы иногда являются пассивными). Коммутаторы могут быть реализованы с одной стороны как распределённые, например в системах с общей памятью, так и сосредоточенные, например в системах ОКМД, а с другой стороны как одноступенчатые, так и многоступенчатые.

Пропускную способность системы чаще всего определяют пассивные компоненты.

15.1 Шинные структуры

Шинной структурой (ШС) называется такая организация системы коммутации, при которой для передачи данных между компонентами вычислительной системы используются одна или несколько общих (а в частном случае и индивидуальных) информационных магистралей. Самым простым случаем организации ШС является *одношинная структура - общая шина*, к которой подсоединены компоненты ВС, как это показано на рис. 15.1.

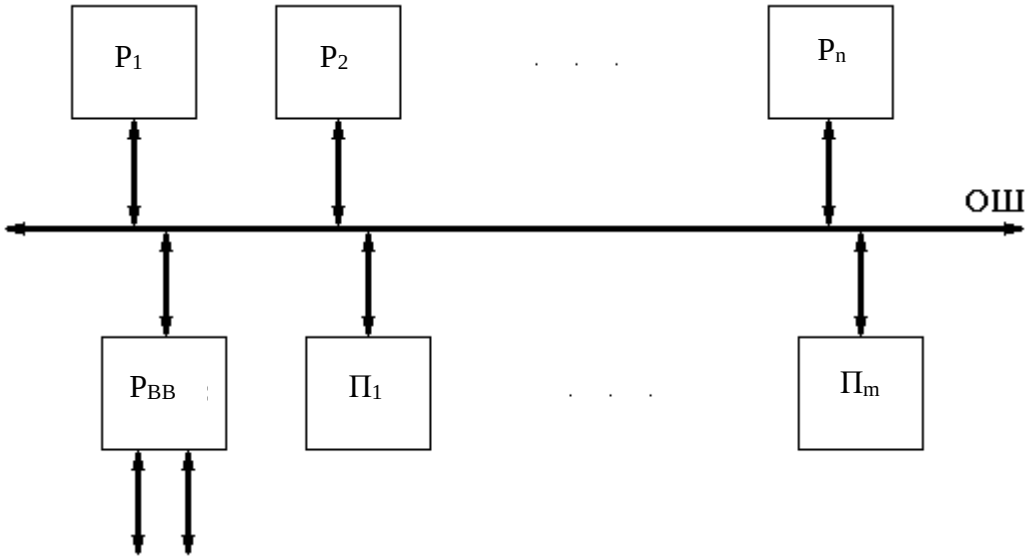


Рис. 15.1 Структура «общая шина»

В этой ВС - n процессоров Р, m модулей памяти П и процессор ввода-вывода РВВ объединены общей шиной ОШ. Системы коммутации, использующие шинные структуры, могут применяться как для ВС с общей памятью, так и для ВС с распределённой памятью. Сами шины представляют собой пассивные элементы, а управлением передачей данных по шинам занимаются либо передающие и принимающие устройства компонент ВС, либо специализированное устройство управления шиной, называемое контроллером шины, при этом часть функций управления остаётся за передающими и принимающими устройствами (адаптеры шины).

Основной трудностью при организации обмена данными между устройствами системы по одной общей магистрали является состязание за право использования общего ресурса - самой шины, которое приводит к возникновению конфликтов при одновременном запросе на передачу данных от нескольких устройств системы. Для разрешения конфликтов

используются такие приёмы, как назначение каждому устройству уникального приоритета, постановка запросов в очередь с различными дисциплинами обслуживания, например. FIFO - "первый пришел, первый вышел", соединение устройств в цепь или в кольцо и, следовательно, физической фиксацией их взаимных приоритетов. Следует отметить однако, что возможна организация логического кольца при физическом подсоединении каждого устройства к общей шине.

Как правило, информационная магистраль (шина) состоит из 3-х групп линий - *линии управления, линии адреса (шина адреса) и линии данных (шина данных)*. При этом в большинстве случаев достаточно 4-х линий управления: линия запроса шины (ЗШ), линия занятости шины (зан. Ш), линия предоставления шины (ПШ) и линия готовности получателя (ГП). Число линий шины данных (ШД) и шины адреса (ША) определяется техническими характеристиками ВС: числом параллельно передаваемых разрядов информационного слова, размером адресного пространства памяти, числом процессоров и другими компонентами системы. На следующем рис. 15.2 представлена типичная схема организации системы коммутации "общая шина".

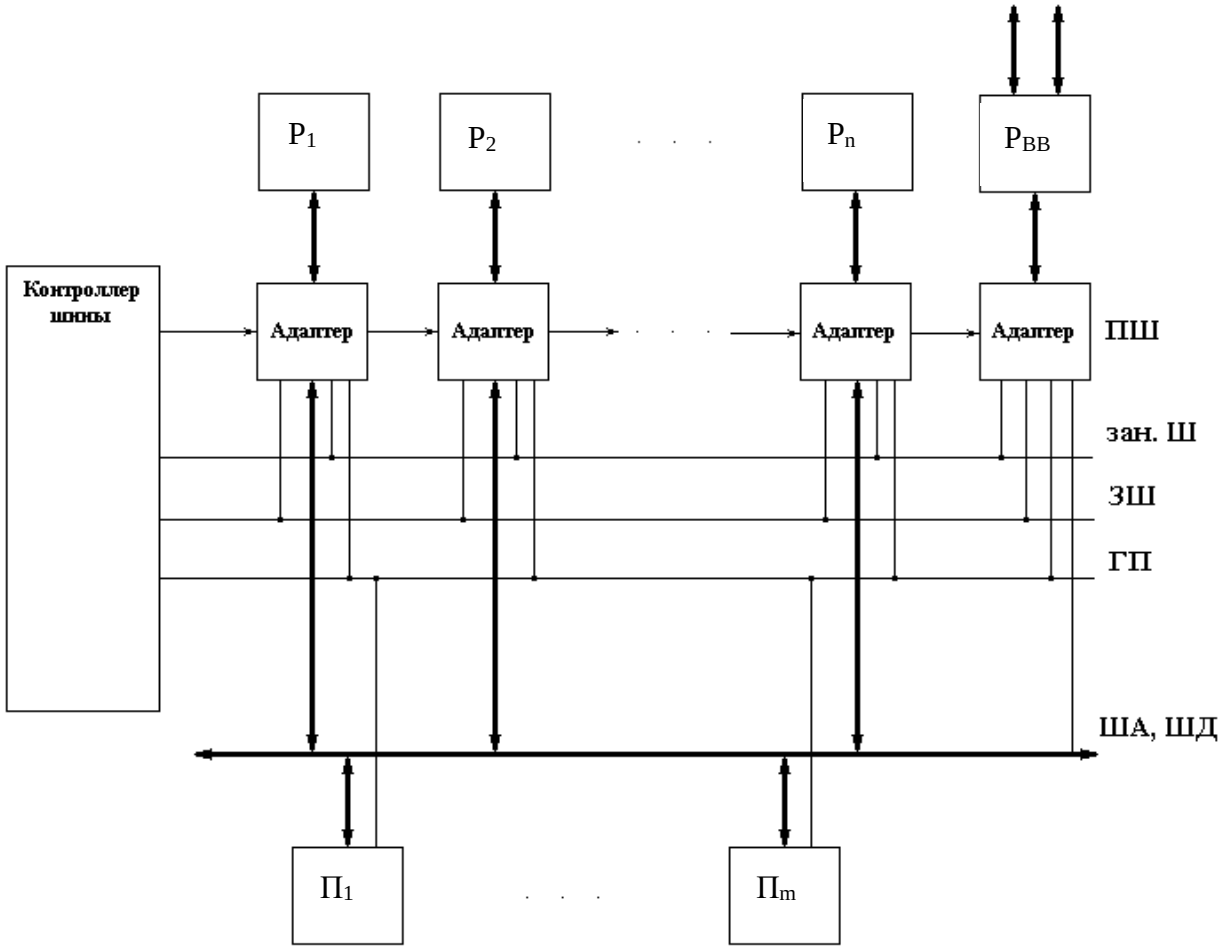


Рис.15.2 Схема организации системы коммутации "общая шина"

Иногда адреса и данные передаются по одной шине с *разделением времени*. В этом случае необходим специальный механизм их различения.

Например, вводится ещё одна линия управления. Функционирование системы коммутации "общая шина" заключается в том, что по заявкам от устройств, желающих передать данные тому или иному адресату, арбитр шины (специальное устройство, входящее в состав контроллера шины при централизованном её управлении) в соответствии с принятой дисциплиной обслуживания при незанятой шине, предоставляет её одному из отправителей данных. После этого отправитель должен установить связь с получателем, который распознаёт свой адрес, установленный отправителем на шине адреса. В случае готовности получателя к приёму данных (о чём он сообщает по линии готовности), отправитель выставляет данные на шине данных, которые и принимаются адресатом. После завершения обмена шина освобождается. Основным недостатком СК с общей шиной является то, что она является общим критическим ресурсом как с точки зрения пропускной способности, так и надёжности СК.

В случае, когда пропускной способности одной шины не хватает для организации обменов данными между компонентами ВС или требуется более высокая надёжность системы коммутации, может быть введено ещё несколько шин. В этом случае каждая компонента должна иметь в своём составе коммутатор, позволяющий соединять передающее и принимающее устройства по одной из свободных шин системы (см. рис. ниже).

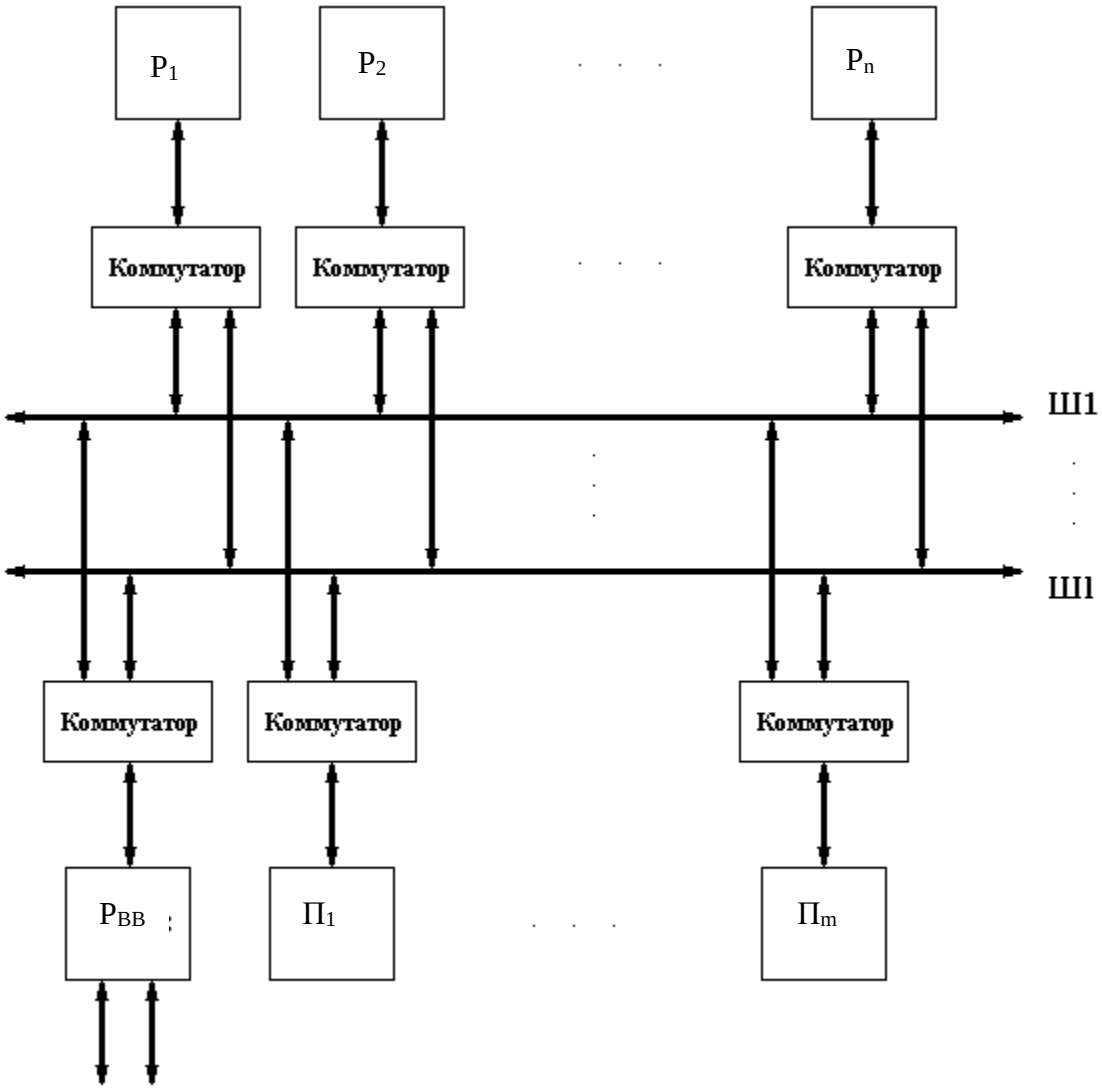


Рис.15.3 Схема организации системы с несколькими шинами

В качестве примера можно привести организацию многошинной системы коммутации многопроцессорного вычислительного комплекса (МВК) "Эльбрус-2" (см. рис. ниже). Данный комплекс относится к классу МКМД с общей памятью. В его состав может входить до 10 центральных процессоров (Р), до 4-х периферийных процессоров (РР) и 8 секций общей памяти (СП).

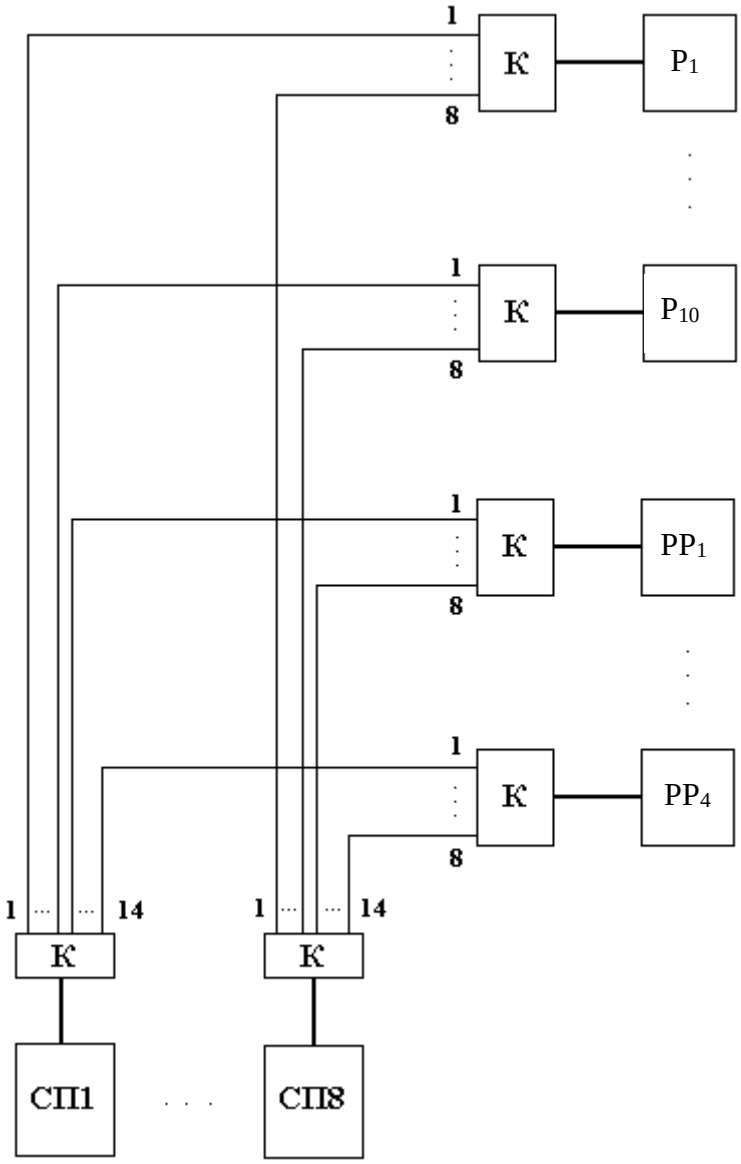


Рис.15.4 Многошинная система коммутации МВК "Эльбрус-2"

Таким образом, каждый из 14 процессоров имеет в своём составе коммутатор на 8 направлений, а каждая секция памяти имеет коммутатор на 14 направлений. При этом система коммутации считается 8-мишинной с распределённым однокаскадным коммутатором. Иногда такую СК называют полносвязной.

Как видно из представленных на рисунках структурных схем, рассматривались ВС с общей памятью. Для систем с распределённой памятью организация СК с шинной структурой с точки зрения построения физического канала принципиально не отличается от такой же СК для систем с общей памятью. Следует однако напомнить, что сам механизм обмена данными в системах с распределённой памятью основан на принципе обмена сообщениями, а в качестве компоненты ВС выступает процессор (или группа

процессоров) со своей локальной памятью, т.е. вычислительный модуль (ВМ). В качестве примера приведём два способа организации полносвязной СК для ВС с распределённой памятью: с распределённым коммутатором - коммутация каждый с каждым (см. рис. ниже)

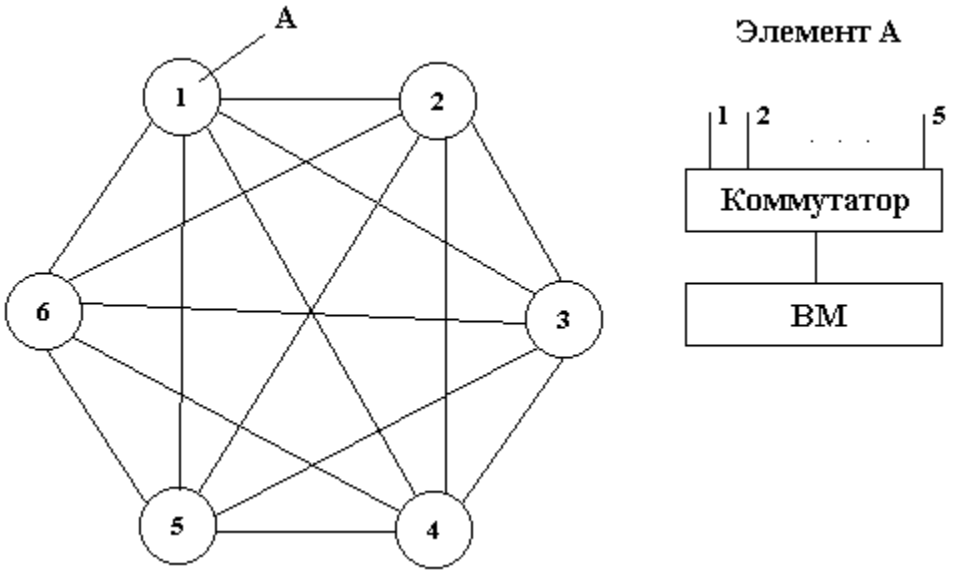


Рис.15.5 Полносвязная СК с распределённым коммутатором (коммутация каждый с каждым)

и центральным коммутатором (ЦК) - соединение типа "звезда" (см. рис. ниже).

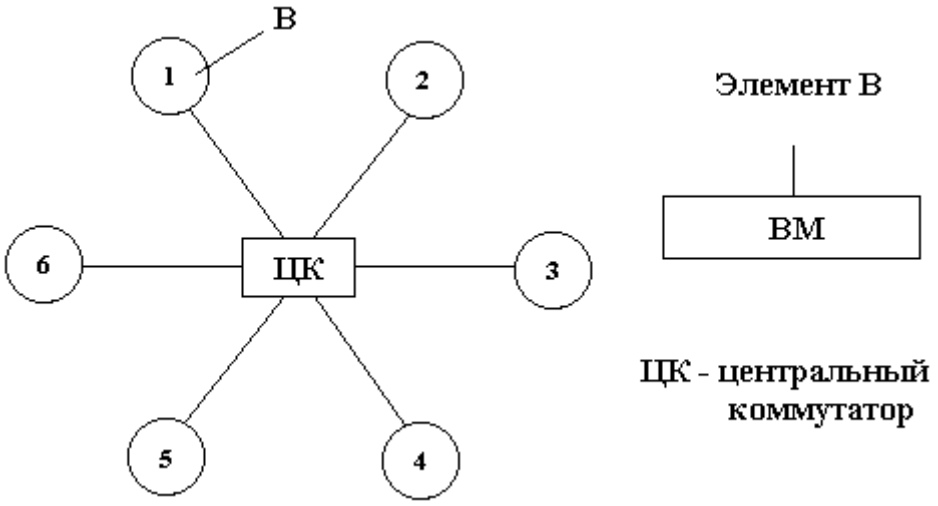


Рис.15.6 Полносвязная СК с центральным коммутатором (соединение типа "звезда")

Первый случай более надёжен, т.к. отсутствует общий ресурс, а второй случай более простой, но менее надёжен из-за присутствия общего ресурса - ЦК.

Рассматривая шинные структуры, приведённые выше, можно сделать один главный вывод - при обеспечении высокой пропускной способности СК вычислительной системы с большим числом процессоров приходится вводить большое число коммутируемых линий связи и сложные коммутаторы, что резко усложняет и повышает стоимость ВС. Поэтому шинные структуры применяются в ВС с небольшим числом процессоров, обычно не более 16-ти.

15.2 Матричные структуры

Системы коммутации матричных структур основаны на использовании множества распределенных одноступенчатых коммутаторов в ВС с высокой степенью распараллельности и с большим числом однородных процессоров. Исторически матричные структуры впервые были использованы в матричных процессорах, относящихся к классу ОКМД. Основное преимущество матричных структур – однородность элементов обработки и регулярность системы коммутации, что дает существенный экономический эффект при реализации матричных процессоров на кристалле. Применение матричных процессоров обычно связано с решением задач цифровой обработки сигналов и изображений и поэтому они рассматриваются как специализированные процессоры в составе мощной высокопроизводительной системы. Базовое соединение элементов процессорной матрицы происходит так, как это показано на рис. 15.7.

Иногда такое соединение называют 2-мерный тор.

Каждый узел данной матрицы представляет собой процессор со своей локальной памятью и коммутатором на 4 направления. Коммутатор может быть как встроенным в процессор, так и внешним по отношению к процессору и может быть реализован в виде специального связного процессора. В матричных структурах обмен данными между соседними процессорами происходит с большой скоростью, которая обуславливается однокаскадным их соединением. Однако, соединение между любыми двумя удалёнными процессорами матрицы требует большего числа каскадов по каналу от источника к приёмнику. В редких случаях, когда требуется высокая пропускная способность СК, применяется так называемая сильносвязная матрица процессоров с магистральным принципом обмена данными, которая изображена на рис. 15.9.

Преимуществом, хотя и дорогостоящим, такой структуры является возможность организации обмена между любыми двумя процессорами, в том числе и в разных строках и столбцах, за один цикл магистрального обмена.

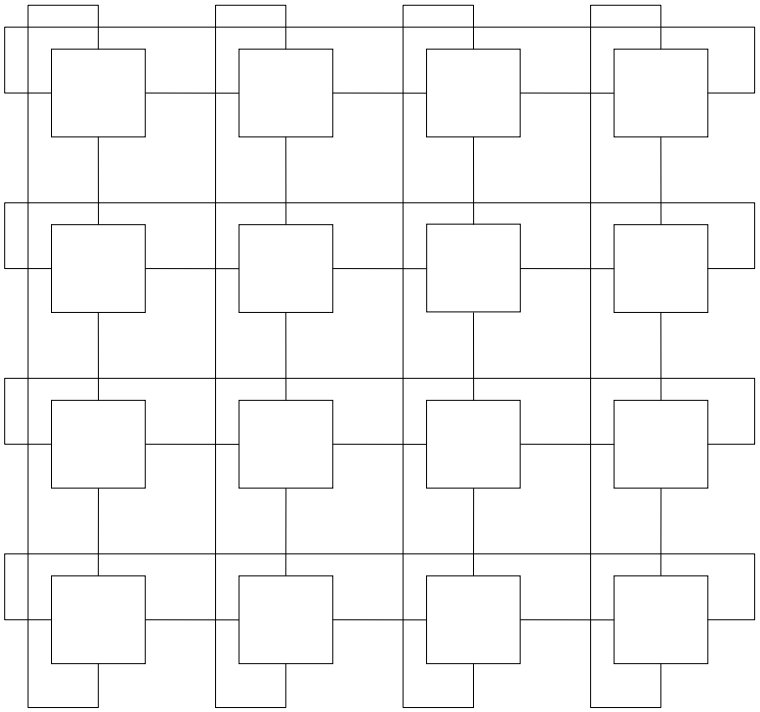


Рис 15.7 2-мерный тор.

Структура элемента матрицы показана ниже, на рис. 15.8.

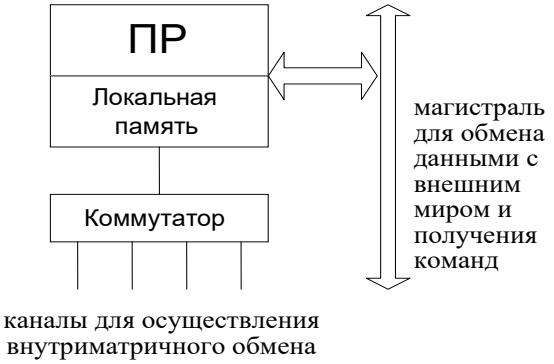


Рис.15.8 Структура элемента матрицы.

Как было отмечено выше, матричные структуры по своей сущности являются специализированными. Но ещё большая специализация достигается в систолических структурах, в которых соединение между процессорами выбираются исходя из особенностей параллельного представления конкретной прикладной задачи, для которой и проектируется данная система обработки данных. (Термин «систолический» заимствован у физиологов. Им обозначается ритмичное сокращение сердца, необходимое для нормальной циркуляции крови в организме животного). Узел систолической структуры, содержащий процессор, ведёт циклическую обработку данных перед тем как

циклически же передать результат обработки одному или нескольким соседям в соответствии с программой решения заданной прикладной задачи.

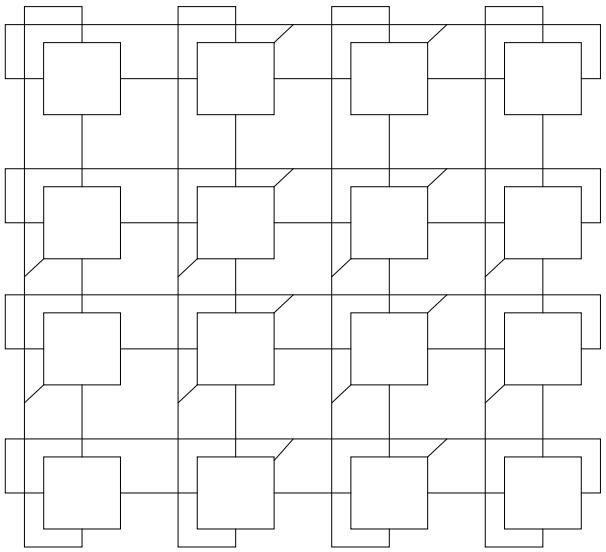


Рис.15.9 Магистральная матрица.

Таким образом, матричные структуры, применяемые, как правило, в системах ОКМД, дают высокие показатели производительности на узком классе задач. Расширение этого класса требует усовершенствования матричных структур. Главной особенностью такого усовершенствования является их применение в ВС, строящихся по принципу МКМД. Например, российская многопроцессорная система МВС-1000 разработана как масштабируемый (легко расширяемый) массив процессорных узлов. Каждый узел содержит процессор с локальной памятью и коммуникационный процессор, имеющий 4 внешних канала (линка). Процессорные узлы связаны между собой по оригинальной схеме, сходной с топологией двумерного тора, представленной на рис. 15.10. Данная структура представляет собой модуль системы, состоящий из 16 узлов. При этом 4 угловых узла матрицы соединяются через линки по диагонали попарно. Оставшиеся 12 линков предназначаются для подсоединения внешних устройств – 4 угловых линка и 8 для соединения с себе подобными модулями Характерной особенностью такого соединения в структурном модуле 4х4 является то, что максимальная длина пути (число каскадов) от одного узла матрицы к другому равна трём.

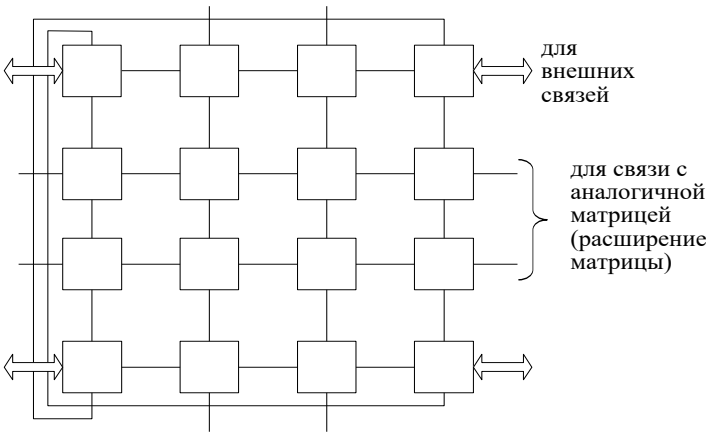


Рис.15.10 Структура МВС-1000.

В качестве примера матричной структуры можно также привести кластер МГУ, структура которого показана на следующем рис. На схеме показан вывод для расширения матрицы. В каждом узле – 2 процессора Pentium III с общей локальной памятью.

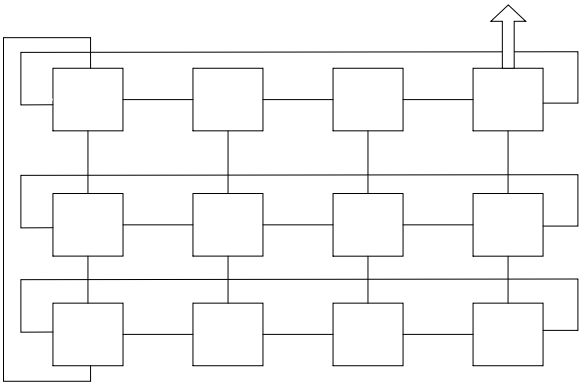


Рис.15.11 Кластер МГУ.

15.3 Кубические структуры

Кубические структуры предназначены для организации ВС с РП, состоящих из большого числа узлов. Они основаны на гиперкубах, организуемых либо в булевском, либо в евклидовом пространстве.

В случае булевского пространства n-мерный куб образуется как декартово пространство кубов меньшей степени. Таким образом, гиперкуб n-го порядка представляет собой два гиперкуба (n-1)-го порядка, соединенных одноименными вершинами.

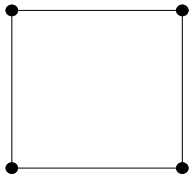
Рассмотрим вычислительную систему состоящую из большого числа вычислительных модулей (ВМ) (процессор со своей локальной памятью). На рис. 15. 12 показана условная схема ВС на примере гипер-куба 4 порядка. В

этой схеме ВМ располагаются в вершинах многомерного куба и соединяются с соседями ребрами этого куба. Для 4-х мерного куба ВС состоит из 16 ВМ (16 вершин), для 12 –и мерного куба из 4096 ВМ (4096 вершин). Соответственно, в первом случае из вершины исходит 4 ребра, во втором – 12 рёбер. Очевидно, что К- мерный куб может включать 2 в степени К ВМ, а максимальный длины путь прохождения сообщений между двумя любыми ВМ составит К рёбер. Рассмотрим более подробно принцип коммутации в схеме соединений типа гипер-куб. Каждый куб следующей размерности строится на основе двух кубов предыдущей размерности соединением двух одноимённых вершин куба предыдущей размерности. Таким образом, каждый раз увеличивая размерность куба на единицу, увеличивается число вершин в 2 раза. Поэтому такой куб называют «булев-К-куб». Булев-К-куб представляет собой очень удобную структуру по следующим причинам.

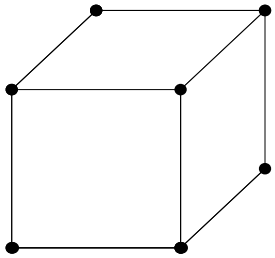
Во-первых, если ВМ располагаются в вершинах 12 –куба, то ни один вычислительный модуль не отстаёт более чем на 12 рёбер куба (т.е. линий связи) ни от какого другого ВМ, что значительно облегчает задачу создания эффективных коммуникаций в системе.

Гиперкубы:
0-го порядка: 1 узел;
1-го порядка: 2 узла;

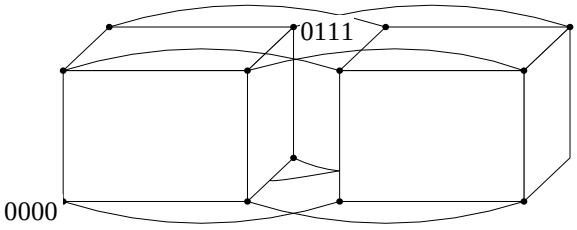
2-го порядка: 4 узла;



3-го порядка: 8 узла;



4-го порядка: 16 узлов;



и т.д.

Рис. 15.12 Принцип образования К-куба

Во-вторых, структура соединений в К-кубе хорошо согласуется с двоичной логикой ЭВМ. Как было отмечено выше, каждый куб в К-кубе имеет два подкуба предыдущей размерности. Если их обозначить значениями булевой переменной 0 и 1, то в результате такого представления каждая вершина имеет единственный, отличный от других, двоичный адрес разрядности К.

В-третьих, одно из ценных свойств К-куба состоит в том, что между любой парой ВМ есть сразу несколько с одинаковым числом рёбер путей коммутации.

Например, если мы имеем дело со структурой 12-куб, то каждая вершина имеет 12 разрядный адрес, первый разряд которого указывает какой из двух 11-кубов содержит данную вершину, второй разряд указывает на соответствующий 10-куб и т.д. до последнего разряда адреса. Этими двоичными адресами можно воспользоваться для того, чтобы направлять сообщение в сеть адресуясь к приёмнику. Получив сообщение, маршрутизатор источника анализирует старший разряд адреса для определения подкуба и, если канал свободен, отправляет сообщение по этому каналу. Если канал занят, маршрутизатор анализирует следующий по старшинству разряд и так до тех пор пока не обнаруживается свободный канал. Так действуют все маршрутизаторы вершин куба по пути от источника к приёмнику.

СМ1- первая система, имеющая процессор кубической архитектуры 12-й степени, состоящий из 4096 узлов ($2^{12} = 4096$). Это система с распределенной памятью. Сколько степеней, столько должно быть и коммутационных выводов у каждого процессора. В этой машине впервые был использован *router* на 12 выходов. ВС такого типа уже имеет сетевую архитектуру. Одним из преимуществ такой схемы коммутации является простота адресации: каждому узлу соответствует определенный 12-разрядный адрес. Каждый разряд соответствует подкубу по старшинству. При передаче данных существует большое количество альтернативных путей. Вычислительный модуль системы показан на следующем рис. 15.13.

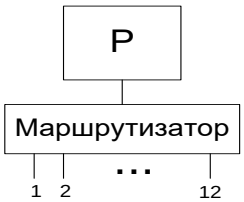
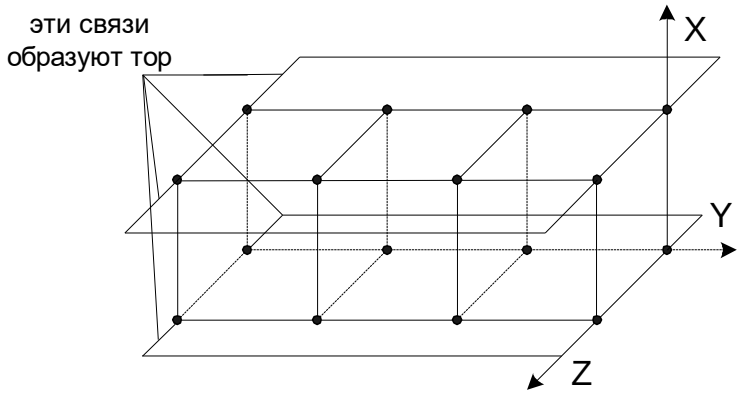


Рис.15.13 Вычислительный модуль системы СМ1.

К кубическим структурам тесно примыкают многомерные торы (в одномерном случае это кольцевая структура, в двумерном – матрица, которые были рассмотрены выше). Торы образуются из n-мерных кубов

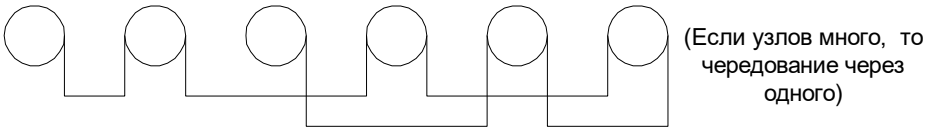
путём введения рёбер между соответствующими вершинами граничных областей по некоторым или всем координатам, как это показано на рис. 15.14. Адресация происходит по координатам. Например, сначала по Y, потом по X и в конце по Z.



Cray T3e/900 serial Number 6702 3-х мерный тор

Рис.15.14 Трёхмерный тор

Используется принцип коммутации каналов. Обычно шина двунаправленная. Для того, чтобы уменьшить значение максимального числа каскадов, соединяющих любые два узла тора, используется так называемое соединения узлов с чередованием.



Соединение узлов с чередованием

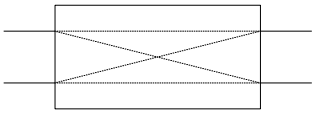
Рис.15.15 Соединение с чередованием

Лекция 13

15.4 Многоступенчатые коммутаторы

При необходимости соединения относительно большого числа (до нескольких десятков) 2N процессоров друг с другом или N процессоров с N модулями памяти и требованием на снижение затрат на систему коммутации по сравнению с полносвязной одноступенчатой коммутацией, применяются многоступенчатые коммутаторы (МК) (чаще всего в ОКМД системах). Время передачи между двумя каждыми элементами ВС должно быть одинаковым. Поэтому число линий связи, соединяющих два элемента из двух множеств тоже должно быть одинаковым.

Основной принцип их построения – создание сложного единого коммутатора с большим числом входов и выходов на основе соответствующим образом объединения простых стандартных коммутаторов. Как правило, МК строятся из коммутаторов 2x2 (рис. 15.16) с двумя входами и двумя выходами, соединяя один из выходов одного коммутатора с одним из входов другого коммутатора (так называемое соединение «точка-точка»). Таким образом, если необходимо построить коммутатор NxN (N входов и N выходов), то число каскадов (ступеней), двух множеств элементов по N элементов в каждом $l = \log_2 N$. Число элементов в каскаде N/2. Таким образом, особенностью такого коммутатора является одинаковое время передачи данных между любой парой элементов ВС. Причём это время пропорционально числу каскадов МК. Обычно коммутаторы 2x2 имеют два состояния – прямое соединение одноимённых входов и выходов и их перекрёстное соединение (рис. 15.16).



Комутатор 2 на 2

Варианты настройки комутатора:

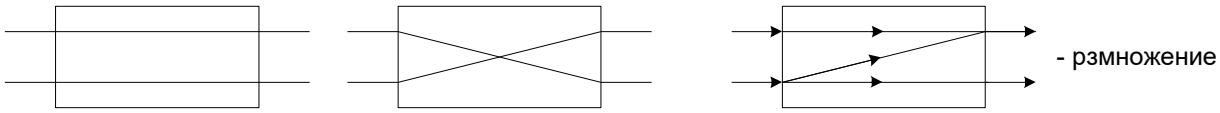


Рис.15.16 Примеры соединений входов и выходов

Состоянием коммутатора необходимо управлять, поэтому каждый из коммутаторов 2x2 имеет свою схему управления, в функции которой входит переключение состояния коммутатора в зависимости от заявок на соединение на каждый из его входов и арбитраж запросов двух входов, если необходимо их соединить с одним и тем же выходом (рис.15.17).

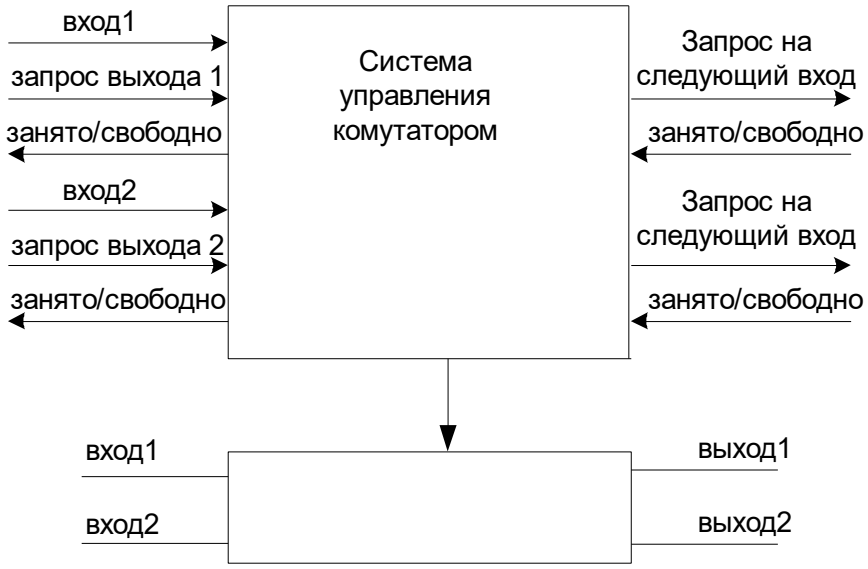


Рис.15.17 Коммутатор со схемой управления

15.5 Сеть Омега - Коммутатор Omega ILLIAC-IV

Сеть Омега была разработана для применения в качестве многоступенчатого коммутатора в ВС матричного типа для параллельной доставки данных из внешней памяти в локальную память процессоров. Соединения входов и выходов в каждой ступени осуществляется по типу «полная тасовка». На рис. 15.8 представлен такой коммутатор 8x8.

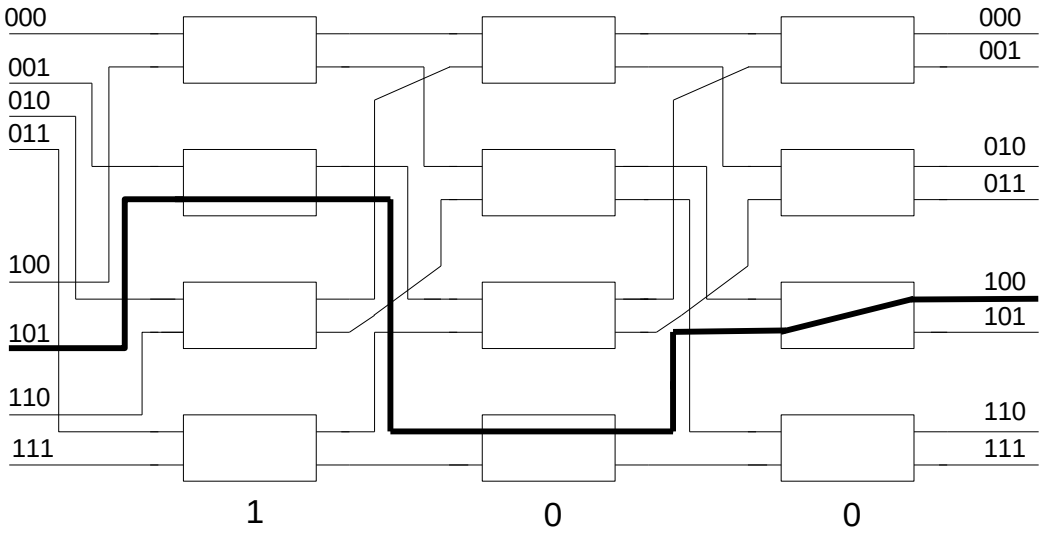


Рис.15.18 Схема коммутатора 8x8

Полная тасовка выполняется так. Первые четыре выхода каждой ступени соединяются с четырьмя верхними входами коммутаторов 2x2 следующей ступени (входы первой ступени соединяются таким же образом с выходами устройств источников запросов). Остальные четыре выхода

ступени соединяются с нижними входами следующей ступени. Таким образом, формируются каналы передачи данных от определённых источников к заданным приёмникам. Для того чтобы реализовать на МК требуемое отображение входов на выходы, необходимо выполнить приведённый выше алгоритм одновременно для всех входов и выходов. При этом некоторые каналы могут быть пересекающимися. В этом случае схема управления коммутатора, в котором возник конфликт, по определённой заранее системе приоритетов предоставляет право на соединение одному из его входов или выходов. Например, высший приоритет назначается верхнему входу коммутатора. В этом случае второй запрос на соединение будет находиться в режиме ожидания освобождения коммутатора.

Существует такой алгоритм управления состоянием коммутаторов. Допустим, что необходимо произвести коммутацию входа S= 101 (s1,s2,s3) с выходом D=100 (d1,d2,d3). Установим коммутатор первой ступени МК, с которым связан вход с номером 101, в состояние соединения этого входа на верхний выход, если d1=0 и на нижний, если d1=1. На следующей ступени коммутации продолжаем действовать по тому же правилу. Вход коммутируем на верхний выход, т.к. d2=0. На последней ступени, в соответствии с тем, что d3=0, коммутатор устанавливает связь с верхним выходом. Следовательно, произошло соединение входа 101 с выходом 100.

Коммутационная сеть Омега относится к системам блокирующего типа, т.е. не всегда можно соединить все восемь входов с восьмью выходами, поскольку необходимо ждать пока нужные коммутаторы разблокируются.

Линии связи могут быть двунаправленными, тогда для коммутации выходов со входами используется тот же алгоритм..

15.6 Коммутационная сеть flip (сеть перестановок)

Коммутационная сеть Flip применяется в машине баз данных STARAN. Принцип соединения – полная тасовка.

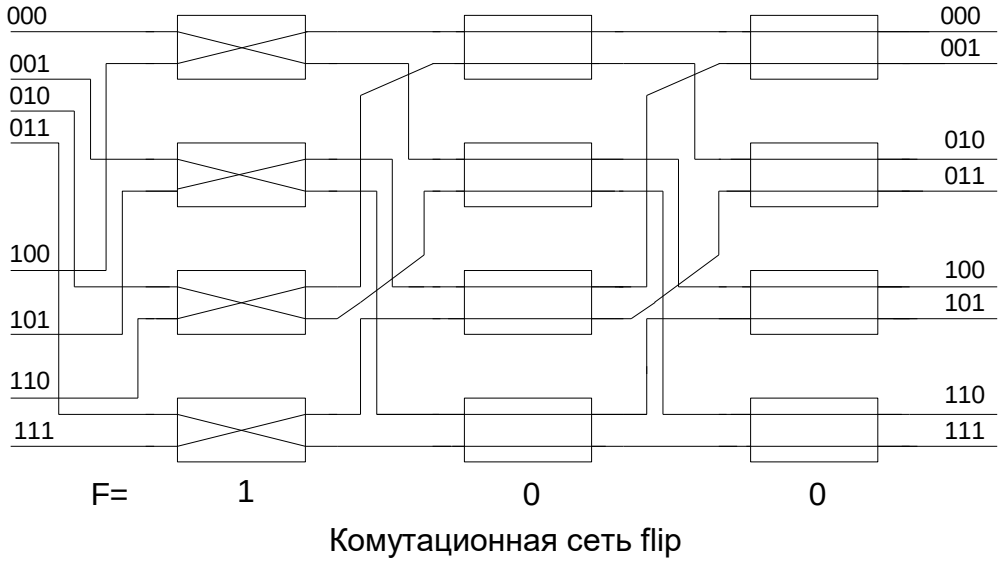
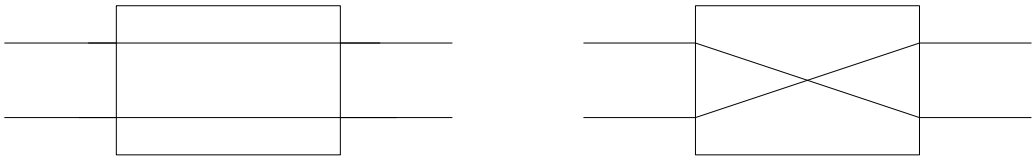


Рис.15.19 Пример настройки сети перестановок

Особенность в том, что сразу настраивается весь «столбец», а не отдельный коммутатор. Состоит из коммутаторов 2 на 2, настраивающихся по следующему принципу:



Всего возможно 8 перестановок, но авторам коммутационной сети flip это показалось недостаточным. Был введён дополнительный управляющий сигнал S (Shift), образующий из столбца циклический регистр. Получается ещё 8 комбинаций.

Итого получилось 64 различных комбинации.

Недостатком системы является то, что эта система блокирующего типа.

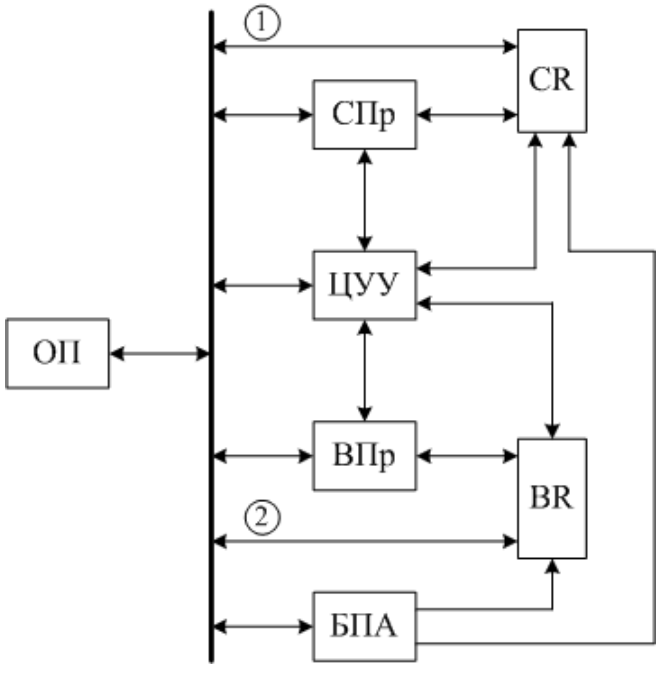
Лекция 14

16. ВС на основе векторных процессоров

Как было отмечено выше, ВС на основе векторных процессоров относятся к классу ОКМД систем. Главный принцип вычислений в такой системе состоит в выполнении некоторой элементарной операции или комбинации из нескольких элементарных операций, которые должны многократно применяться к разным данным. Такие операции отображают векторные команды, которые входят в состав системы команд векторного процессора и операндами которых являются вектора. Таким образом, модель вычислений векторного процессора – одиночная операция выполняется над большим блоком данных. Обычное проявление этой вычислительной модели в исходной программе – циклы на элементах массива, в которых значения, вырабатываемые на одной итерации цикла, не используются на другой итерации цикла. Основой выполнения векторной команды является конвейер операций. Так как при выполнении векторной команды одна и та же операция применяется ко всем элементам вектора (или чаще всего к соответствующим элементам пары векторов), для настройки конвейера на выполнение конкретной операции может потребоваться некоторое установочное время, однако затем операнды могут поступать в конвейер с максимальной скоростью, допускаемой возможностями памяти [7].

Обычно такие ВС называют супер-ЭВМ; минимальная конфигурация – один векторный и один скалярный процессор. Обычно это системы с общей памятью.

СПр – скалярный процессор
ВПр – векторный процессор
БПА – блок подготовки адресов
ЦУУ – центральное УУ
ОП – общая память
CR – набор скалярных регистров
BR – набор векторных регистров



Система команд:

1. скалярные команды;
2. векторные команды.

Минимальное число векторных регистров – 8; длина слова – 64 разряда. Возможно выполнение операций над полусловами (32 разряда) и двойными словами (128 разрядов). Обработкой более мелких единиц информации (байты и т.д.) занимается скалярный процессор.

Длина одного векторного регистра – 64×64 разряда (64 операнда, длина каждого операнда – слово). Изначально векторные регистры были однопортовыми, потом им на смену пришли двупортовые – регистры, из которых можно считывать загруженные данные, не дожидаясь окончания загрузки (записи) всего регистра! Обратное недопустимо!

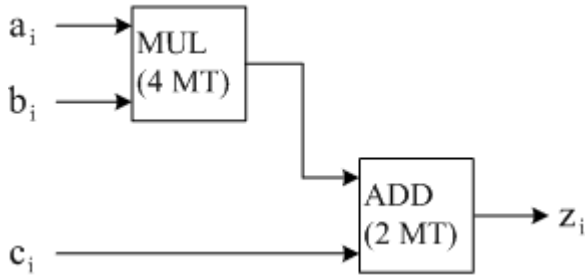
Особенность: можно одной векторной командой загрузить векторный регистр, одной командой суммировать содержимое регистров и одной командой сохранить результат в памяти, т.е. ***C := A + B*** – за 4 команды!

Принцип зацепления. Если для каждой операции выделить автономное специализированное исполнительное устройство (ИУ), использующее принцип *конвейера операций*, и если последовательность векторных команд такова, что результат q-й команды используется в $(q+1)$-й, то этот результат можно подавать непосредственно с выхода ИУ, выделенного для q-й операции, на вход ИУ, выделенного для $(q+1)$-й операции.

Под *конвейером операций* здесь понимается следующее: если стадию E выполнения операций, время исполнения которой больше одного МТ, разбить на этапы, равные одному МТ, и выделить на исполнение каждого этапа автономный функциональный узел, то эту стадию можно конвейеризировать. Таким образом, если выполняется конвейер операций умножения, то первый результат умножения двух элементов векторов появится, например, через 4 МТ (это время называется *латентным периодом*), а последующие результаты будут появляться через 1 МТ!

Пример. ***Z := A×B + C***

Пусть размерность векторов – 64 элемента. Тогда первый результат (z_1) будет получен через 6 тактов, а все последующие – с интервалом в один такт. Вся операция над векторами будет выполнена за $(6 + 1\times63)$ тактов!



Отметим, что такой выигрыш в производительности на векторных процессорах можно получить только на операциях над векторами; скалярные данные и вектора малых размерностей выгоднее обрабатывать на скалярных процессорах.

Принципы скалярной обработки. Командное слово i считывается из памяти команд в регистр команд скалярного процессора. В адресном поле A командного слова содержится указание на данные j. Одновременно со считыванием данных j из памяти данных, считываются данные k из регистра, заданного в поле R командного слова i. Над данными j и k а АЛУ выполняется операция, задаваемая в поле КОП командного слова. Результаты операции заносятся в регистр R. В этом примере данные j, представляющие собой единичные элементы обработки и размещённые в

памяти независимо друг от друга, называются скалярными. Команды, инициирующие операции над скалярными данными, называются скалярными командами. Процессор, выполняющий скалярные команды, называется скалярным процессором. Схема скалярной обработки приведена на рисунке ниже.

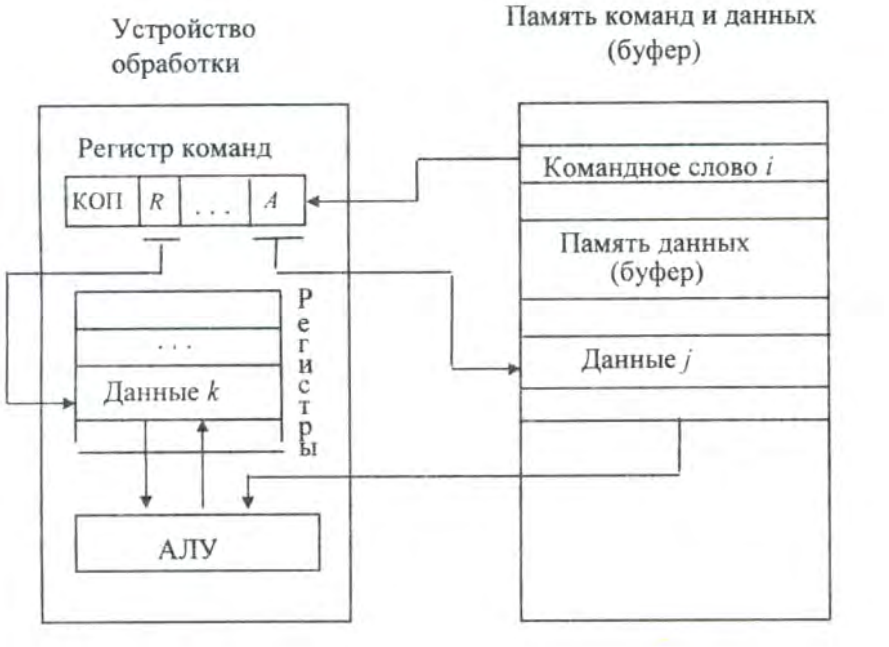


Рис.16.1 Схема скалярной обработки

Принципы векторной обработки. Командное слово *i* считывается из памяти команд в регистр команд векторного процессора. Отличительной особенностью векторного процессора ос скалярного процессора является наличие в его системе команд векторных команд и аппаратных средств поддержки их реализации.

Очевидно, что векторные команды предназначены для выполнения операций над векторными данными, которые представляют собой *e*-элементные наборы чисел, размещённые в памяти с определённой регулярностью и над которыми должны выполняться одни и те же действия. Векторные данные имеют особо важное значение для реализации высокоскоростной числовой обработки, т.к. с помощью одной команды можно указать операцию над множеством данных, а в процессоре приводить в действие различные механизмы параллельной обработки. Указание командным словом векторных данных в векторных процессорах осуществляется по-разному, но при этом прямо или косвенно должны быть заданы:

1. адрес *A* базы векторных данных;
2. значение приращения адреса *d* между элементами векторных данных;
3. число элементов и длина вектора *e*.

Для записи считываемых из памяти векторных данных процессор должен иметь регистры, ёмкостью достаточной для хранения данных вектора длиной *e*. Эти регистры называются векторными. Следует обратить внимание на то, что устройства, ведущие непосредственную обработку данных (АЛУ), могут быть в СПр и ВПр одинаковыми.

Таким образом, при векторной обработке предварительные действия перед непосредственным выполнением операций требуют большого объёма работ, связаны с начальной установкой различных регистров (длина вектора – *e*, смещение *d*, которое может быть переменным для элементов вектора).

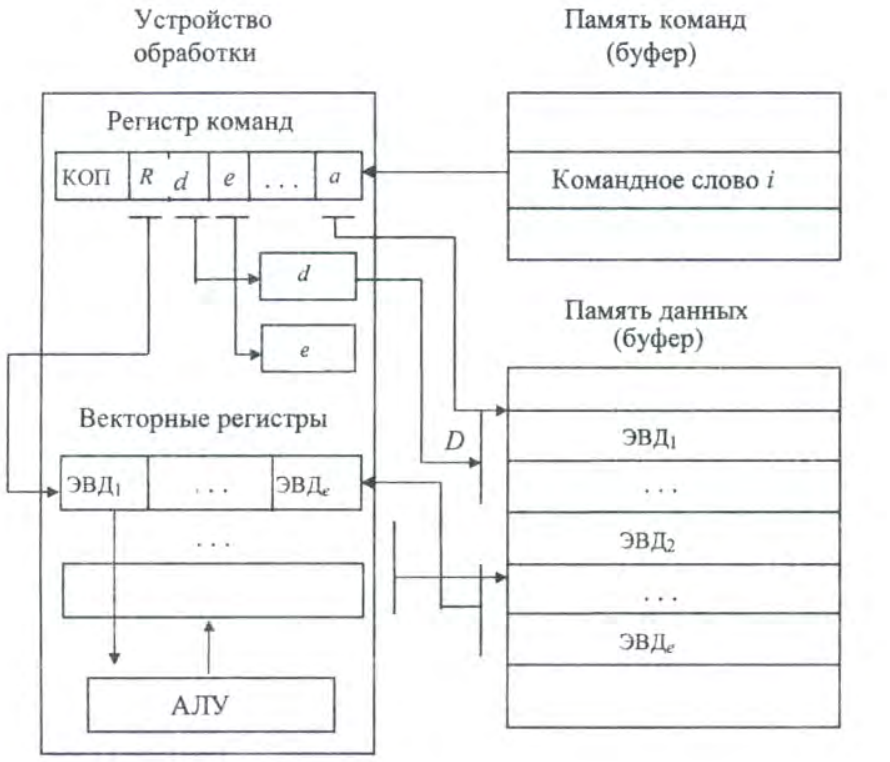


Рис.16.2 Схема векторной обработки

Однако эти предварительные действия не предваряют выполнения всего цикла, как это происходит при скалярной обработке, и рассредоточены в стадиях выполнения векторных команд.

Преимуществом же векторной обработки перед скалярной является то, что для выполнения одной и той же операции над *e* элементами вектора выбирается из памяти векторная команда один раз. Следовательно, именно этот фактор даёт возможность одной командой инициировать *e*-кратное выполнение одной операции над различными данными. Таким образом, появляется принципиальная возможность эффективного использования конвейерных исполнительных устройств процессора.

Производительность ВПр возможно увеличить за счёт:

1. введения многомодульной памяти с возможностью параллельной выборки операндов;
2. применение конвейерных исполнительных устройств (конвейер операций);
3. введение нескольких конвейерных исполнительных устройств (либо одинаковых, либо специализированных для отдельных операций);
4. организация принципа зацепления операций.

Лекция 15

17. ВС класса ОКМД на основе матричных процессоров

Исторически понятие «матричный процессор» возникло при создании многопроцессорной вычислительной системы, предназначенной для решения задач, в которых присутствует большая доля операций над матрицами. При этом структура физических связей между процессорами (т.к. эти процессоры первоначально были достаточно простыми, они получили название «процессорный элемент») не обязательно соответствует матричным соединениям. В настоящее время матричные процессоры (МП) применяются для решения таких задач как цифровая обработка сигналов, обработка изображений, ядерной физики и т. д. В качестве примера можно привести задачу обработки изображений в реальном масштабе времени с алгоритмами опознавания объекта и проверки соответствия его геометрических и физических свойств заранее заданным характеристикам. Такая прикладная задача обычно в качестве составляющей части включает анализ кадра изображения, например, состоящего из 1024x1024 пикселей, со скоростью 1000 кадров в секунду. Это означает, что за секунду необходимо обработать около миллиарда пикселей изображения. Если, например, анализ каждого пикселя требует 10 операций, то суммарное быстродействие системы должно превышать 10 миллиардов операций в секунду. Так как любая задача содержит, как правило, и другие части с меньшим параллелизмом, но требующих тоже высокого быстродействия, то МП обычно является составной частью (приставкой) мощной системы обработки данных.

Структура такой СОД содержит обычно следующие составляющие.

Главная вычислительная машина (ГВМ).

Интерфейсная система, связывающая ГВМ с МП.

Коммутационная система, связывающая компоненты МП друг с другом и периферийными устройствами.

Матричный процессор, обычно состоящий из большого числа ПЭ, например 8x8.

ВС на основе матричных процессоров строится как система с распределенной памятью (каждый ПЭ имеет локальную память) в составе СОД с общей памятью, структура которой представлена на рис. 17.1.Условные обозначения на этом рисунке.

1 – Мощная коммутационная сеть с буферами и коммутаторами.

2 – Матрица процессорных элементов (матричный процессор). Все элементы работают синхронно – это ОКМД в чистом виде. Обычно матричный процессор состоит из элементов с сокращённым набором команд

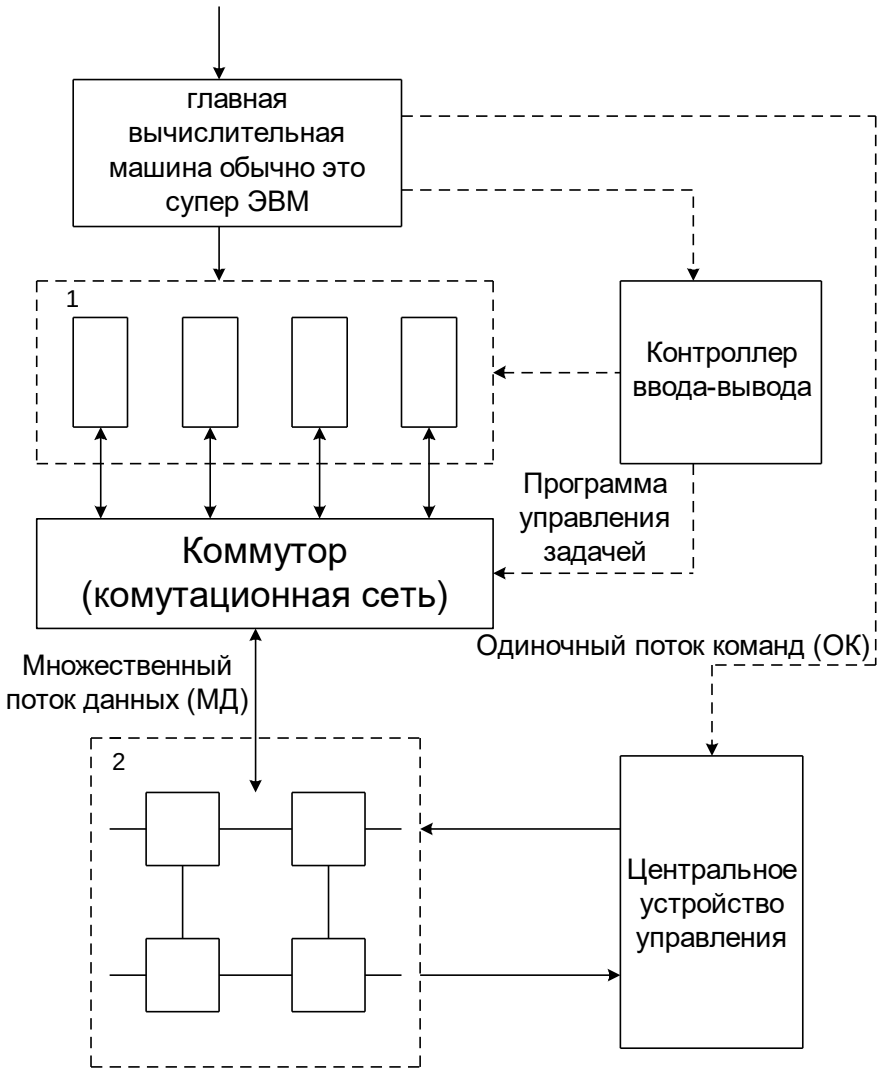


Рис. 17.1 Структурная схема ВС на основе матричного процессора

.

Функции ЦУУ (Центрального Устройства Управления):

1. Дешифрация команды и её рассылка всем процессорным элементам матрицы. Все ПЭ выполняют одну и ту же команду со своими данными.
2. Генерация адресов и данных и передача их процессорным элементам.
3. Особенность матричных процессоров – т.к. на все процессорные элементы поступает одна команда, то и адрес, по которому надо записать (считать) данные тоже один. Поэтому при выполнении команды, связанной с доступом к памяти, в этой команде указывается только один адрес, но каждый ПЭ, обладая индексным регистром, модифицирует его и обращается к своей локальной памяти по своему адресу.
4. Данные и программа поступают в локальную память каждого процессора. Но программа одна, поэтому если она большая, то она распределяется между локальными памятями процессорных элементов.

Центральное управляющее устройство обеспечивает считывание команд из локальной памяти процессорных элементов в нужном порядке.

5. Организация связи между процессорными элементами и ПЭ с внешней памятью системы..
6. Приём и обработка сигналов прерываний от ПЭ, устройств ввода-вывода и ГВМ.

Связь между ПЭ друг с другом осуществляется синхронно через сеть связи (матрица), используя механизмы коммутации каналов. Так, если i-му ПЭ нужны данные, хранящиеся в памяти j-го ПЭ, то необходимо сначала проложить маршрут (скоммутировать канал) между ними, а затем передать данные. Очевидно, что для организации параллельных вычислений на множестве ПЭ, одновременно должно коммутироваться множество каналов, что должно быть отражено в самой прикладной программе. Именно это является одной из основных трудностей параллельного программирования для ВС, построенных на основе матричных процессоров.

Основной же трудностью организации связи между ПЭ и внешней памятью, является параллельная доставка в локальные памяти множества ПЭ различных данных.

Имеет место проблема условных переходов: пока одна ветвь программы работает, процессоры другой ветви стоят. Получается, что если программа богата разветвлениями, то часть процессорных элементов постоянно простаивает.

17.1 Структурная схема ILLIAC- IV

Б – буферы

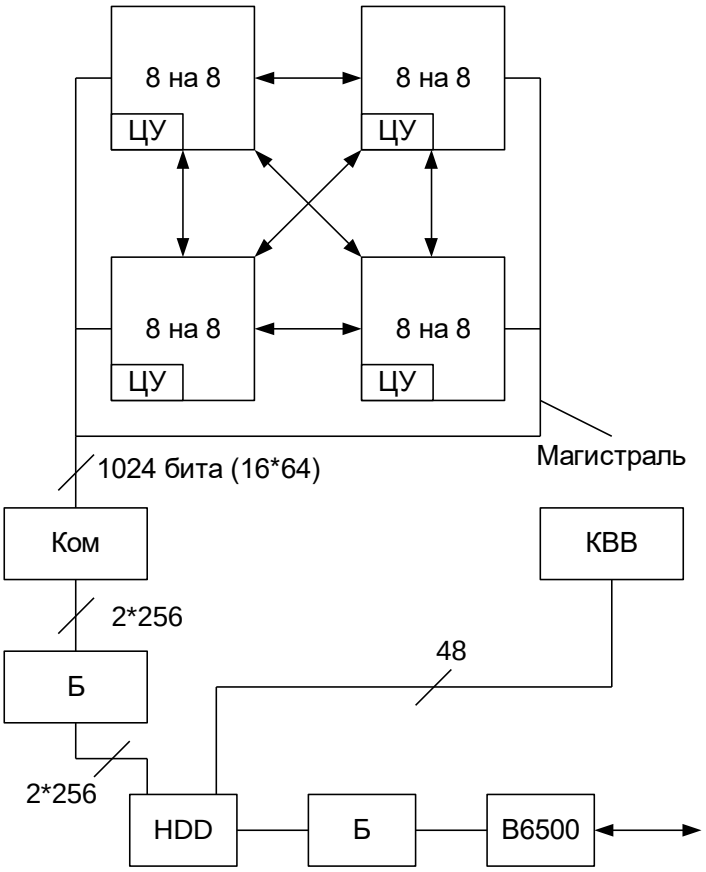
В6500 –вычислительная машина, на которой функционирует операционная система, транслятор (подготовка программного кода, который будет исполняться на данной ВС).

ЦУ – центральное устройство управления (своё в каждом процессоре).

КВВ – контроллер ввода-вывода.

Ком – коммутаторы.

Оказалось что магистраль (для подкачки операндов) оказалась настолько узким местом, что использовался только один (!) матричный процессор. Поэтому реально выпускали систему с одним матричным процессором (8 на 8).



Структурная схема ILLIAC - IV

Пример: перемножение матрицы X и Y размером 4*4, результат в матрицу Z.

Реализация на ILLIAC-IV.

Каждый процессорный элемент имеет свои АЛУ и регистры. Пусть A и B-регистры под операнды, а S – регистр для хранения промежуточных результатов.

Из матрицы 8*8 нам достаточно 4, т.е. возьмём столбец процессоров.

Распределение памяти для хранения элементов этих матриц в локальной памяти процессорных элементов представлено в таблице.

Используются следующие обозначения - ПЭ1, ПЭ2, ПЭ3, ПЭ4- 4 процессорных элемента матрицы процессоров, к – номер ячейки локальной памяти.

Следует обратить внимание, что графически матрицы представлены по столбцам, а логически по строкам.

	X	Y	Z
пэ \ к	11 12 13 14	21 22 23 24	31 32 33 34
ПЭ1	x11 x21 x31 x41	y11 y21 y31 y41	z11 z21 z31 z41
ПЭ2	x12 x22 x32 x42	y12 y22 y32 y42	z12 z22 z32 z42
ПЭ3	x13 x23 x33 x43	y13 y23 y33 y43	z13 z23 z33 z43
ПЭ4	x14 x24 x34 x44	y14 y24 y34 y44	z14 z24 z34 z44

Замечание: размещение элементов матрицы в памяти – отдельная проблема, которая решается отдельно для каждой задачи.

Программа, реализующая рассматриваемый алгоритм, может быть описана следующим способом:

Организуется внешний цикл i от 1 до 4 (по числу строк матрицы X) и внутренний цикл j от 1 до 4 (по числу строк матрицы).

- i = 1 первая строка матрицы X передаётся в ЦУУ (все ПЭ выполняют одну и туже команду, в результате чего содержимое ячеек с номером к=11 оказывается в ЦУУ(т.е. x11, x12, x13, x14)).
- j = 1 из ЦУУ значение x11 передаётся в А-регистры всех ПЭ.
- третья команда. Все ПЭ выполняют команду загрузки из своей ЛП в регистр В содержимого ячейки к =21. Таким образом, в В - регистрах четырёх ПЭ оказываются соответственно y11, y12, y13, y14.
- Перемножить содержимого регистров А и В на всех ПЭ.
- Прибавление результатов умножения к содержимому S-регистров (у каждого процессора свой регистр S, т.е. адреса регистров одинаковые, но находятся они в разных процессорах).
- …….
- После завершения внутреннего цикла в регистрах S всех ПЭ оказывается значение элементов первой строки результирующей матрицы z11, z12, z13, z14. Эти значения переписываются в ЛП в ячейки с номером к =31.
- Изменяя значение i, запоминаются последовательно все строки матрицы Z, значение каждого элемента, который высчитывается по формуле:

$$Z_{ij} = \sum_{l=1}^{4} x_{il} * y_{lj}$$

Заметим, что параллельное выполнение алгоритма возможно в том случае, если элементы матриц пересылаются из различных ПЭ в ЦУУ параллельно, а сами элементы строк матриц размещены в ЛП процессорных элементов в ячейках памяти с одинаковыми адресами. При выполнении различных операций с матрицами задача размещения их элементов в памяти является довольно сложной. Для облегчения этой задачи в ряде МП используется специальный механизм индексации адресов локальной памяти в каждом ПЭ.

Рассмотренный простой пример раскрывает возможность параллельной работы ПЭ, организованной в соответствии функционирования ВС типа ОКМД, не затрагивая вопросы обмена данными между ПЭ. Однако ту же задачу на том же матричном процессоре можно реализовать таким образом, что одни ПЭ выполняют операции умножения, другие и умножение и сложение. В этом случае обмен между ПЭ неизбежен. Таким образом, составляя параллельный алгоритм и его программную реализацию на данном типе ВС, необходимо оценить различные варианты и выбрать тот, который даёт наименьшее время выполнения данной прикладной задачи.

17.2 Нисходящее проектирование матричных процессоров

Как было отмечено выше, МП представляет собой специализированную приставку к ВС высокой производительности. Следовательно, должен создаваться как вычислитель, эффективно выполняющий узкий класс заранее известных прикладных задач. Основные требования, которые предъявляют к МП при его проектировании следующие:

1. МП представляет собой регулярный массив ПЭ, выполняющих на протяжении каждого машинного такта одинаковые вычислительные операции с пересылкой результатов вычислений своим соседям.
2. Соединительные линии между ПЭ должны быть короткими.
3. ПЭ должны эффективно выполнять основные (базовые) операции, характерные для заданного класса прикладных задач.
4. Должно быть установлено оптимальное соотношение между ALU процессорного элемента и пропускной способностью памяти.
5. Наличие соответствующего языка параллельного программирования.

Таким образом, если задана прикладная задача и поставлена цель создания МП, эффективно его реализующего, необходимо сначала разработать параллельный алгоритм, ориентированный на реализацию с использованием МП, а затем решить проблему формализованного перехода от описания алгоритма к структуре МП. При этом описание параллельного алгоритма должно быть с одной стороны таким, чтобы было просто для понимания пользователями, а с другой стороны было бы исходным материалом для системы автоматизированного проектирования , которая осуществляет компиляцию в эффективную структуру процессорной матрицы на СБИС.

На рис. 17.2 представлена схема так называемого нисходящего проектирования процессорной матрицы, которое начинается с формализованного описания алгоритма с учётом присущих ему свойств рекурсии и параллелизма и заканчивается описанием аппаратной реализации конфигурации СБИС.

Нисходящее проектирование означает разбиение процесса проектирования на этапы с помощью перехода от описания алгоритма к структуре матрицы.

Целевая функция:

1. Определение процессорного элемента, т.е. определение состава команд и процесса обработки (последовательный или параллельный).
2. Выбор такой топологии, которая позволила бы для данной конкретной задачи минимизировать число линий связи, задействованных в решении этой задачи.

Рис.17.2 Схема нисходящего проектирования матричного процессора

Распишем алгоритм нисходящего проектирования матричных процессоров более подробно:

Этап 1. Есть некоторое функциональное представление задачи (Завершается построение алгоритма).
Шаг1. Представление алгоритма в параллельном виде. (В основном выполняется вручную).
Шаг2. Выделение базовых операций (этот шаг и все следующие в основном выполняются автоматически).
Шаг3. Булевы функции для каждой из базовых операций.

Этап2. Функциональный компилятор осуществляет отображение исходного алгоритма в структурно-функциональное описание процессорной матрицы. Т.е. создаётся архитектура процессорной матрицы.
Шаг4. Взаимосвязи процессорных элементов. Связи обычно задаются жёстко, т.к. конечный продукт – это приставка к более сложной ВС, которая обычно выполняет какую-нибудь определённую функцию, например цифровую обработку видео сигнала.
Шаг5. Сам процессорный элемент.
Шаг6. Уровень побитной обработки данных (последовательной или параллельной).

Этап3. Кремниевый компилятор отображает структурно-функциональное описание в топологию СБИС.
Шаг7. Система топологических связей на пластине.
Шаг8. Топологический рисунок на пластине (включая процессорные элементы).
Шаг9. Уровень топологических ячеек. Представляет собой заключительный вариант топологии, которая будет выращена на кристалле.

Свойства полученной структуры:
1. Короткие линии связи между процессорными элементами, что позволяет повысить пропускную способность.
2. Регулярность самой структуры (одинаковые процессорные элементы и линии связи). Это позволяет увеличить плотность упаковки на кристалле и упростить его производство.
3. Высокая степень распараллеливания вычислений.

Факторы, на которые при разработке матричного процессора надо обращать особое внимание:

- Разработчик должен обеспечить быстрое выполнение базовых функций – это его основная задача.
- Нужно учитывать принцип конвейеризации. Каждый процессорный элемент должен совершать операции в конвейерном режиме.
- Нужно выбрать оптимальное соотношение между производительностью процессора и пропускной способностью памяти.
- Должны создаваться специальные языки параллельного программирования, ориентированные на матричные процессоры.

Пример: машина ПС2000 (создана для управления атомными электростанциями). (см. рис. 17.3)

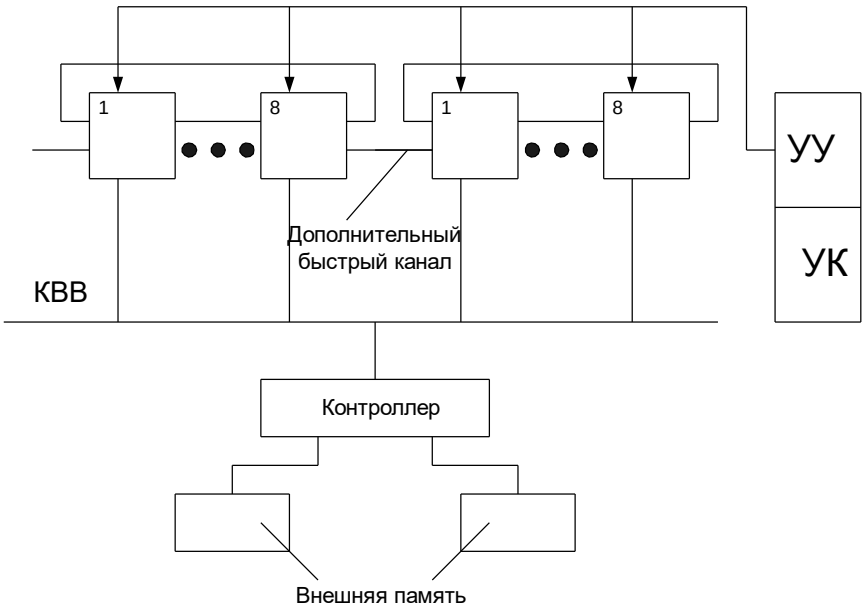


Рис.17.3 Структурная схем ПС-2000

По сути – это матричная система.

Идеологически система строится как матрица 8 на 8, но процессорные элементы здесь гораздо мощнее, чем в ILLIAC-IV.

Конструктивно – это совокупность линеек из восьми процессорных элементов. Может использоваться от одной до восьми линеек. Линейки процессорных элементов соединяются специальным быстрым каналом связи. Каждый процессор имеет свою локальную память.

Особенности:

1. Одна из первых в мире масштабируемых систем.
2. В устройстве управления (УУ) впервые применено ПЗУ базовых микрокоманд и загружаемая память микрокоманд. т.е. система имеет возможность перепрограммирования для выполнения другой совокупности микрокоманд. Замечание: в системах, называемых «адаптивными» адаптация чаще всего происходит именно так.
3. УК – управляющий комплекс, основная функция – подготовка задач для реализации в матричной системе и её мониторинг.

Пример: CM-1 (connection machine) фирмы thinking machines.

В основе лежит гиперкуб двенадцатой степени (4096 узлов).
Система строилась для обработки трёхмерных изображений.
Конструктивно выполнена из 8 подкубов.
Каждый узел состоит из 16 процессорных элементов.
Каждый процессорный элемент может работать автономно.
Каждый процессорный элемент имеет одно одноразрядное АЛУ, которое может выполнять всего 4 операции и 7 одноразрядных регистров.
Каждый процессор обрабатывает одну точку изображения.

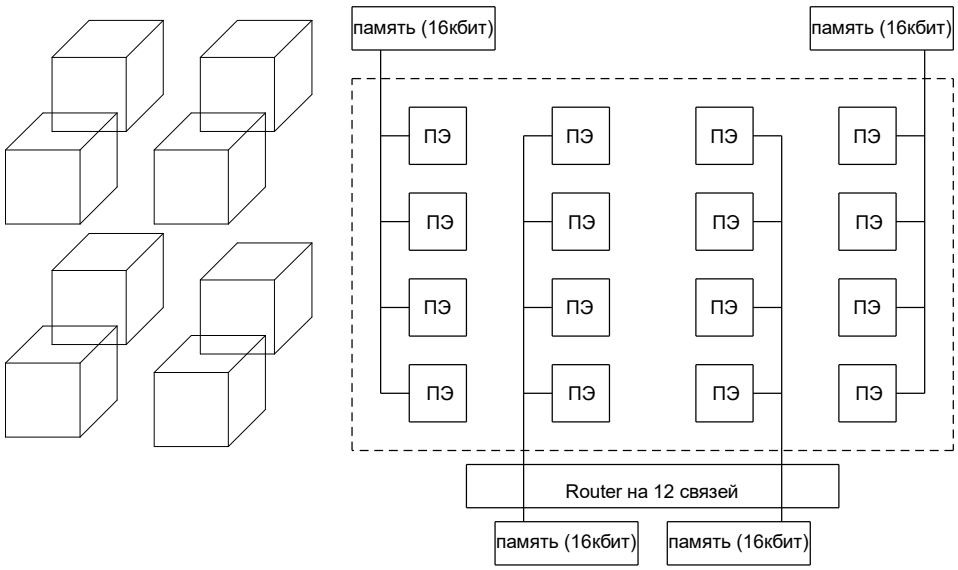


Рис.17.4 Схема узла СМ-1

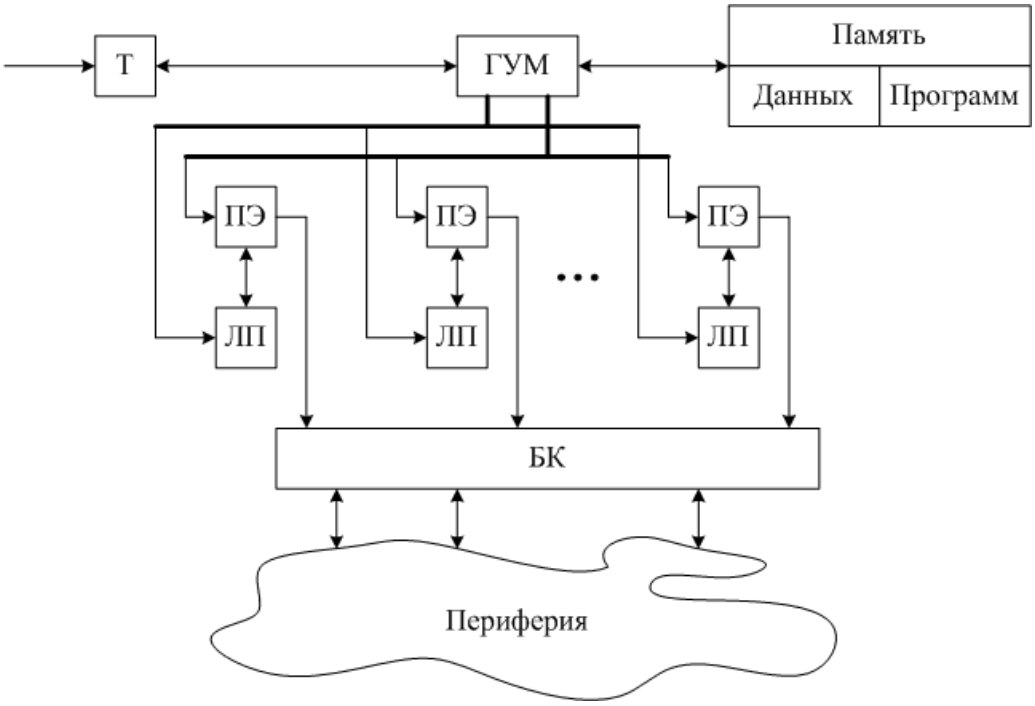


Рис. 17.5 Структурная схема СМ-1

ГУМ – главная управляющая машина
ПЭ – процессорные элементы, 64К штук, соединены гиперкубом
ЛП – локальная память, 4К
БК – быстрые коммуникации
Т – терминал, через который оператор управляет ВС

17.3 Оценка эффективности ВС на основе матричных процессоров

Синхронная параллельная обработка элементов массивов может выполняться на матричных процессорах типа *ILLIAC*-4 или векторных процессорах типа *CRAY* или *CYBER*-205 [6].

Системы типа *ILLIAC*-4 в своё время получили распространение, и они действительно давали очень высокое быстродействие на определённом классе задач, например, быстрого преобразования Фурье и задач математической физики. С другой стороны, для многих задач применение процессоров такого типа не приносит требуемого эффекта, хотя задачи хорошо распараллеливаются. Для них выгодно применять векторные процессоры. Поясним это на примере. Обозначим через p число ПЭ в системе типа *ILLIAC*-4, а через T_P – время, необходимое для выполнения вычислений при использовании p процессоров. При этом под T_P будем понимать не физическое время, которое аналитически определить трудно, а число "шагов вычисления". Если последовательность вычислений представить в виде дерева, то шаг вычислений будет соответствовать ярусу дерева. Среднее ускорение за счет применения p процессорных элементов определим из соотношения:

$S_P = \frac{T_1}{T_P}$ – среднее ускорение за счет применения p ПЭ

$E_P = \frac{S_P}{P}$ – степень полноты загрузки p ПЭ.

Определим эффективность применения такого типа процессоров на примере перемножения матриц. Число параллельных операций, выполняемых при перемножении векторов, равно числу процессорных элементов p, которые могут быть задействованы при вычислении вектора.

$N_C = \left[\frac{n}{P}\right]$ – число шагов необходимое для вычисления вектора-результата, состоящего из n элементов; $[x]$ – ближайшее целое, большее x.

$N_O = [log_2 n] + 1$ – число операций сложения элементов вектора, полученного в результате умножения двух векторов, при условии, что число ПЭ достаточно для проведения вычислений (пересылки между ПЭ и другие вспомогательные операции не учитываются).

Определим ускорение вычислений произведения двух матриц размерности $l \times n$ и $n \times q$ с использованием векторных операций и параллельного вычисления сумм элементов векторов для 64 процессоров. Пусть $n = q = l = 64$.

На однопроцессорной машине для вычисления двух матриц надо выполнить $2 \times l \times q \times n$ операций. На параллельной машине с p процессорами надо выполнить $l \times q \times [n/p]$ векторных умножений и $l \times q \times ([log_2 n] + 1)$ сложений (при условии $n \le p$).

Ускорение при выполнении вычислений определяется соотношением:

$$S_P = \frac{2lpq}{lq\left(\frac{n}{P} + \left(\left[\log_2 n\right] + 1\right)\right)}$$

Для $p = 64$ и $n = 64$ $\quad S_{64} \sim 18 \div 19 \quad$ $E_{64} \sim 0{,}3$

Таким образом, даже в одном из самых благоприятных для системы случаев производительность 64 процессорных элементов оказалось равной только 19. Надо отметить, что процессорные элементы *ILLIAC*-4 достаточно дорогие, поскольку выполняют некоторые функции устройств управления и имеют собственную относительно большую память.

Вообще же даже для наиболее хороших задач ситуация оказывается далеко не такой благоприятной, как в этом примере. В результате анализа некоторого числа задач была получена статистика, показанная в таблице.

№ группы программ	Процент скалярных операций		Процент векторных операций
	Не выполняемых параллельно	Выполняемых параллельно	
1	4	27	69
2	8	0	92
3	9	18	73
4	11	3	86

Операции, показанные в столбце 3, которые можно выполнить параллельно, в большинства случаев разнородные. Например, сложение и умножение на матричном процессоре могут быть выполнены только последовательно (не могут быть совмещены по времени).

Процент скалярных операций определяется суммой столбцов 2 и 3 таблицы. Для группы задач:

№1	$\beta_1 \approx 31\ \%$	$S_{64} \approx 3$	$E_{64} \approx 0{,}05$
№2	$\beta_1 \approx 8\ \%$	$S_{64} \approx 11$	$E_{64} \approx 0{,}2$
№3	$\beta_1 \approx 27\ \%$	$S_{64} \approx 3{,}5$	$E_{64} \approx 0{,}05$
№4	$\beta_1 \approx 14\ \%$	$S_{64} \approx 6{,}1$	$E_{64} \approx 0{,}09$

Отсюда видно, какой вес в снижении производительности матричных процессоров играет даже относительно небольшой процент скалярных операций.

Указанные свойства вычислительных систем типа *ILLIAC*-4 позволяют рассматривать их как проблемно-ориентированные процессоры, показывающие высокую производительность при решении ряда важных задач статистической обработки и математической физики. Большое внимание в последнее время уделяется векторным процессорам. Одна из их особенностей – введение в процессор векторных команд.

Векторные команды и векторное задание операндов позволяют

эффективно организовать магистральную обработку данных и легко рассчитывать адреса элементов вектора в процессе выполнения векторной команды. Использование векторной команды вместо циклического выполнения скалярных команд резко сокращает число обращений к оперативной памяти за командами и экономит время на обработку команд.
Интересным является векторный процессор *BSP* (*Burroughs Scientific Processor*) с быстродействием до 500 млн. операций, с плавающей запятой в секунду. Процессор *BSP* рассматривается как векторный процессор, расширяющий возможности высокопроизводительных вычислительных систем *B*7800/*B*7700. В нем имеется 16 процессорных элементов, которые работают под общим управлением, но связаны с общей для всех процессорных элементов памятью с параллельным доступом, кроме этого, есть мощный процессор для обработки скалярных команд.

Параллельная обработка скалярных величин резко повышает эффективность распараллеливания. Процессор для скалярных вычислений может обрабатывать параллельно 3 операции и 16 процессорных элементов обрабатывают векторные команды.

Используя данные таблицы, получим следующие оценки эффективности выполнения программ для группы задач:

№1	$S_{19} \approx 6$	$E_{19} \approx 0{,}32$
№2	$S_{19} \approx 7{,}6$	$E_{19} \approx 0{,}4$
№3	$S_{19} \approx 5{,}3$	$E_{19} \approx 0{,}28$
№4	$S_{19} \approx 3{,}9$	$E_{19} \approx 0{,}21$

Т.е. для этих задач эффективность использования аппаратуры на векторных процессорах типа BSP или CRAY значительно выше, чем на системах типа *ILLIAC*-4.

Лекция 16

18. Супер-ЭВМ семейства CRAY (ВС с ОП)

Появление векторных процессоров обусловлено наличием прикладных задач, в составе которых в значительной части присутствуют операции над векторами и матрицами большой размерности (явный параллелизм) или обработка множества циклов. Несмотря на то, что в последнее время появились высокопроизводительные скалярные (суперскалярные) процессоры, имеющие в своём составе несколько ядер, каждый из которых имеет параллельно функционирующие исполнительные узлы, а команды выбираются из памяти в виде связок, представляющих собой длинное слово (содержащее, например три команды, которые могут быть выполнены параллельно), их производительность и свойства оказываются востребованными.

В 1976 году под руководством Сеймура Р. Крея была создана первая супер-ЭВМ CRAY-1, идеология векторной обработки которой используется до сих пор [7].

CRAY-1 – впервые применена векторная идеология (векторно-конвейерная). Была применена водяная система охлаждения: сначала использовалась дистилированная вода, позднее – вода с теплоёмкими добавками, ещё позднее – газ. Процессор вместе с памятью занимал ~6 м^2, кэш-память отсутствовала. Базовая модель – однопроцессорная ВС.

$T_{CY\,P}$ = 12,5 нс (*CRAY*-1)

$T_{CY\,P}$ = 4,1 нс (*CRAY*-2)

Модели *CRAY*-3 и *CRAY*-4 были заявлены, но не были построены из-за дороговизны изготовления и сложности технологических процессов (кристалл необходимо было выращивать в космосе). В настоящее время не существует даже макетов упомянутых моделей.

CRAY-XMP – идеология *CRAY*-1, но ВС многопроцессорная. Число процессоров: 8, 16, 32…

Базовая ОС – ОС *Unix*, базовый язык программирования – *Fortran* 77.

Структурная схема ВС *CRAY*-1 на базе одного процессора приведена на рис. 18.1. *Команда* трёхадресная: *<1-й операнд>, <2-й операнд>, <результат>*

В буферы команд (всего их 4) за один машинный такт записывается 4 слова памяти. Одновременно может работать 12 функциональных устройств, (впервые был применён принцип зацепления), разделённых на 4 группы.

Группа адресных исполнительных устройств.

1. сложение целых чисел (2 такта);
2. умножение целых чисел(6 тактов).

Исполнительные устройства подготавливают адреса операндов; непосредственного участия в решении прикладных задач не принимают.

Группа скалярных операций.

1. сложение целых чисел (3 такта);
2. логические операции (1 такт);
3. сдвиг (1÷2 такта);
4. счётчик (2 такта); возможен счёт по модулю.

Группа операций с плавающей точкой.

1. Сложение (6 тактов);
2. Умножение(7 тактов);
3. вычисление обратной величины (эквивалент деления) (14 тактов).

Группа операций над векторами.

1. сложение чисел (3 такта);
2. логические (2 такта);
3. сдвиг (4 такта).

Вектора целочисленные. Если элементы вектора – действительные числа, то обработка идёт на блоках 3. Все функциональные устройства полностью сегментированы. Это означает, что в каждый такт на вход устройства может поступать новый операнд (или набор операндов), не связанный с предыдущим, при этом в устройстве может продолжаться обработка предыдущих операндов.

Все системы CRAY имеют в своём составе множество различных регистров, в том числе, адресные регистры (A), скалярные (S), промежуточные адресные (B), промежуточные скалярные (T), векторные (V) и большое число вспомогательных регистров.

CRAY-2 строилась как многопроцессорная.

Общая память. Общая оперативная память (ОП), построенная на МОП-транзисторах, содержит порядка 256 млн. слов. ОП состоит из 128 блоков с чередующимися адресами по 2 млн. слов в каждом. Как и в других машинах семейства *CRAY*, слово содержит 64 информационных и 8 контрольных разрядов.

Произвольный доступ к ОП имеют любой из четырёх процессоров нижнего уровня и любой из быстрых каналов и каналов общих данных. Все операции доступа к памяти выполняются автоматически на аппаратном уровне. Любая программа пользователя может обращаться как ко всей памяти, так и к любой её части.

Общий поток обмена информацией с памятью – 64 Гбайт/с.

Локальная память. Ёмкость LM в каждом процессоре нижнего уровня составляет 16384 64-разрядных слова. Эта сверхбыстродействующая память служит для хранения скалярных операндов в течение всего времени счёта и для временного хранения элементов векторов, когда они используются для вычислений с помощью векторных регистров более одного раза. LM обменивается данными с адресными регистрами, а также со скалярными и векторными регистрами, заменяя тем самым регистры *B* и *T*, существовавшие в ранних моделях систем.

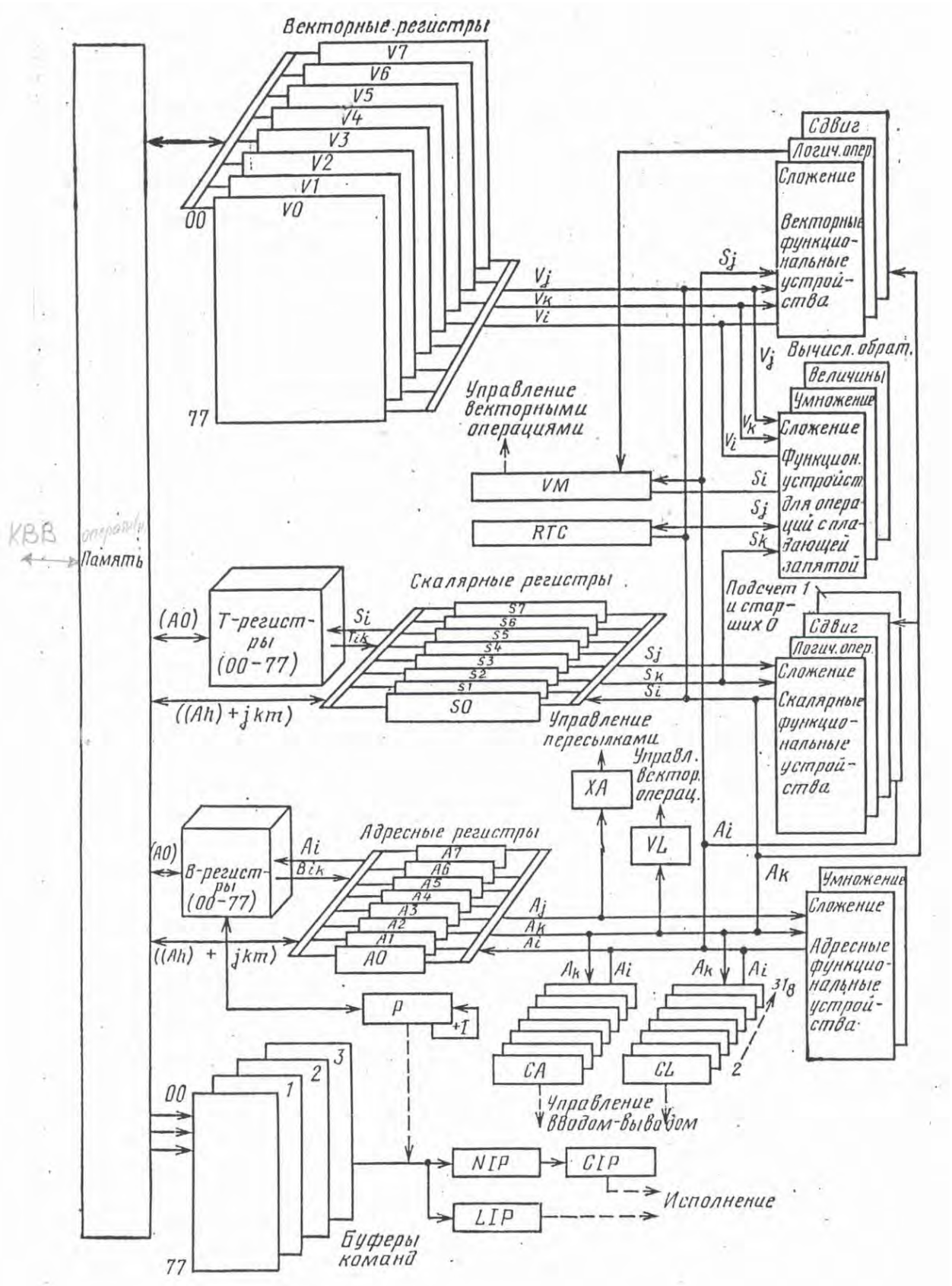


Рис. 18.1 Структурная схема вычислительной системы *CRAY*-1

Время передачи данных из LM в регистры *A* и *S* или в обратном направлении составляет один машинный такт. Начальное заполнение векторного регистра занимает пять тактов. После завершения заполнения данные могут поступать в регистр в каждом такте. Обращения в LM могут перекрываться с обращениями к ОП.

Обмен с памятью осуществляется страницами. Работать можно с одной страницей, а параллельно заполнять другую. Используется принцип старения.

Каждый процессор нижнего уровня выполняет арифметические и логические операции. Эти операции, так же как и прочие функции процессора нижнего уровня, координируются устройством управления.

Резюме:

CRAY-2 – отличается от *CRAY*-1 тем, что блоки *T* и *B* заменены на локальную память. Также есть отличия в функциональных блоках.

Существует четыре процессора нижнего уровня (минимум), причём на четыре процессора нижнего уровня есть один процессор верхнего уровня.

18.1 CRAY C-90

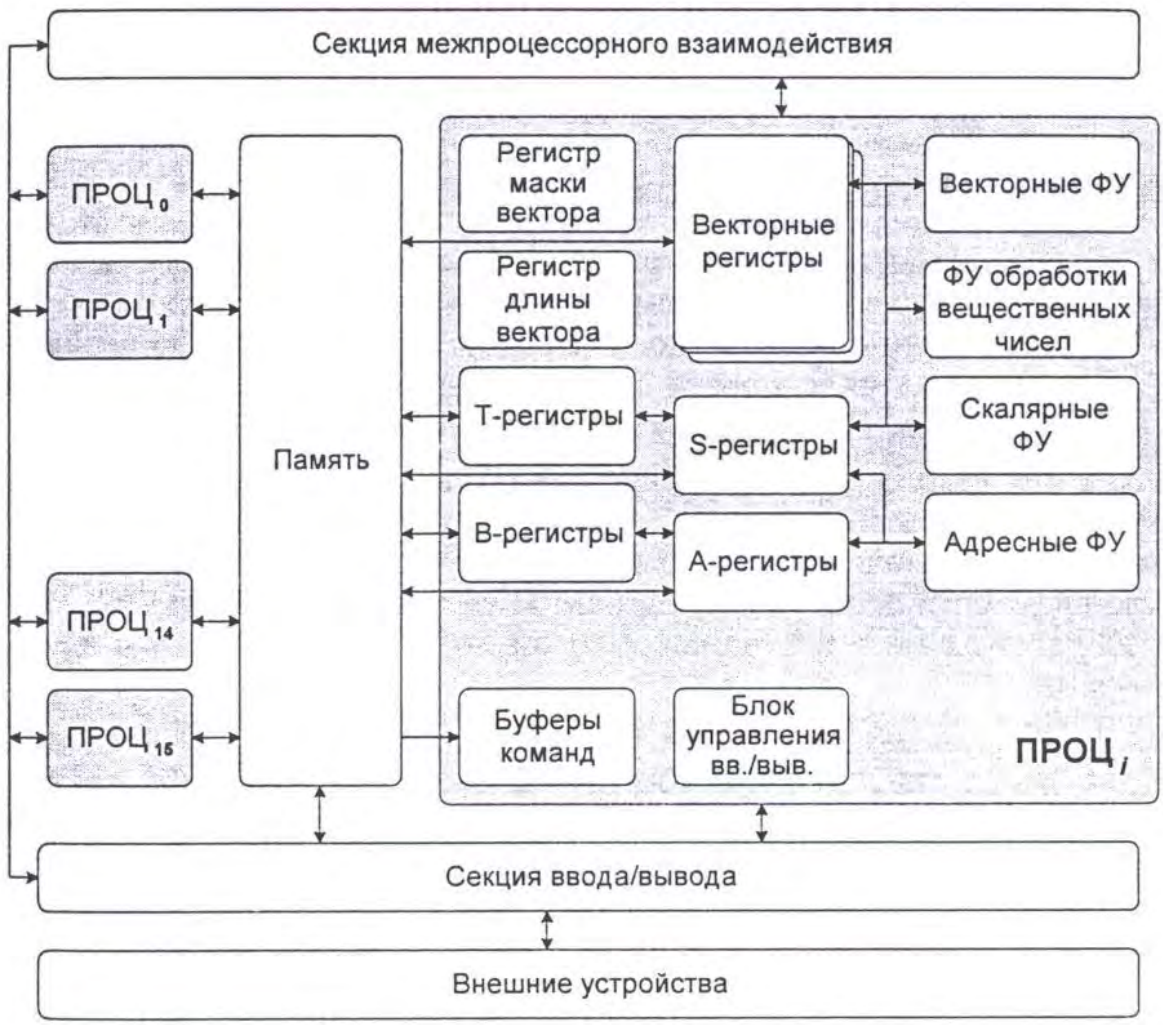


Рис.18.2 Общая схема компьютера *CRAY C*-90

Блоки «Векторные ФУ» и «ФУ обработки вещественных чисел» имеют по *два конвейера*! (впервые в *CRAY* применена идеология *NEC*)

Векторные ФУ (5–7 штук):

1. сложение/вычитание;
2. сдвиг;
3. логические;
4. умножение битовых матриц и т.д.

ФУ обработки вещественных чисел:

1. сложение/вычитание;
2. умножение;
3. обратная величина (эквивалент операции деления).

Скалярные ФУ:

1. сложение/вычитание;
2. умножение;
3. логические;
4. сдвиг.

Адресные ФУ:

1. сложение;
2. умножение.

Ёмкость буфера команд: 8 блоков × 16 (32) слов – одна страница.

Пример. ***C := A + B*** На рис. 18.3 изображён принцип конвейерного выполнения операций над элементами вектора, разделённых на чётные и нечётные, с одними работает один сумматор (+), с другими другой.

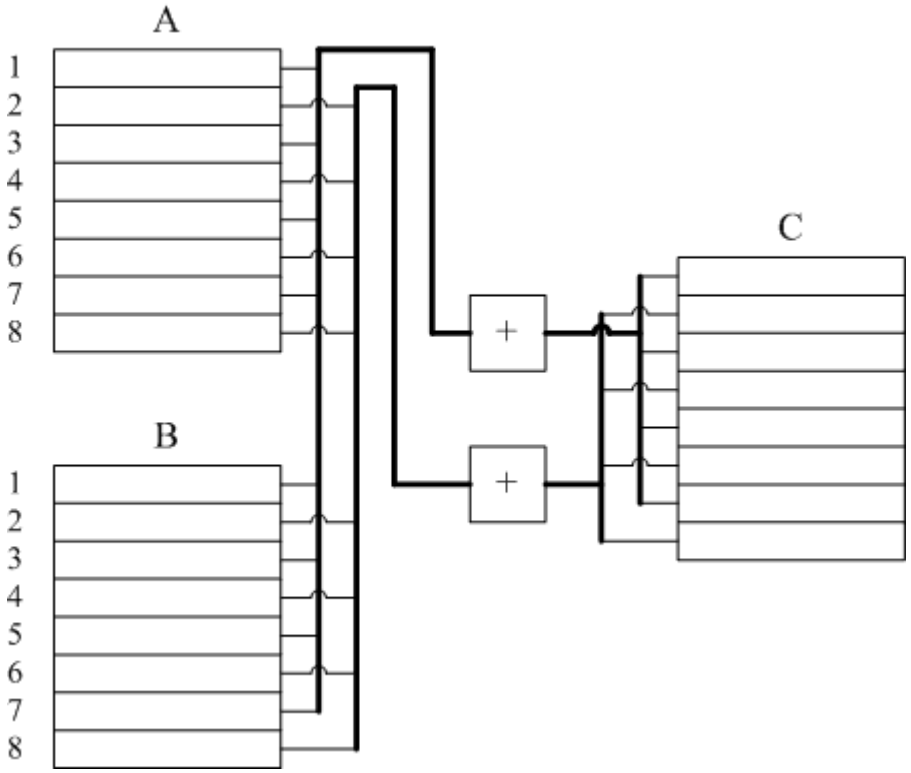


Рис. 18.3 Выполнение операции над векторами

Благодаря этому производительность по сравнению с *CRAY*-1 выросла в два раза! В скалярном процессоре это бессмысленно, ведь каждый раз надо подавать команду, а здесь – ОКМД.

Оценка производительности. Рассмотрим операции с вещественными числами (ведь они самые длинные). $T_{CY.\ P.\ CRAY\text{-}2} = T_{CY.\ P.\ CRAY\ C\text{-}90} = 4{,}1$ нс. Для трёх ФУ получаем $T = 4{,}1 / 3 = 1{,}3$ нс – это для одного процессора. Всего 16 процессоров, следовательно, $T = 1{,}3 / 16 = 0{,}08125$ нс, т.е. $f = 12{,}3$ ГГц!

Разрядность слова – 64 разряда.

Имеется 16 исполнительных устройств.

ОП имеет 16×4 портов и разделена на 8 секций, а каждая секция – на 8 подсекций, в каждой подсекции 16 банков. Получаем 8×8×16 расслоений памяти. Адреса в секцию записываются так: 0→0, 1→1, …, 7→7, 8→0…

Если длина вектора кратна степени двойки, то возможны конфликты между процессорами, т.к. два процессора могут обратиться к одной секции! Конфликт в секции памяти устраняется за один такт, в подсекции – за 6 тактов, в банке – несколько десятков тактов!

В настоящее время это ограничение накладывает весьма значительные ограничения на архитектуру ВС – пока не научились делать эффективные ВС с числом процессоров, больше 16-и (из-за конфликтов памяти)!

Зависимость производительности от длины вектора здесь имеет нелинейный характер…

18.2 СуперЭВМ NEC SX (1÷6)

ВС с ОП *NEC SX*-6 обладает самой высокой производительностью!

T_C = 1 нс (реально, скорее всего, немного больше).

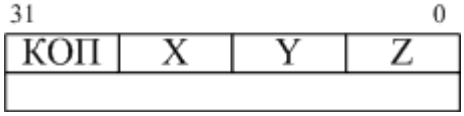
ВС состоит из 128 узлов, в каждом из которых 8 процессоров. Эта машина относится к классу 3Т (терабайтные КС, ёмкость памяти – сотни терабайт, $f \sim$ ТГц). Внутри порядка 40 векторных регистров, которые играют роль кэш-памяти.

Система может работать в режимах:

1. пакетной обработки;
2. случайного потока задач с любых терминалов;
3. предполагается удалённый доступ.

После того, как программа введена, управляющий процессор компилирует её и выполняет привязку входных данных к заданию; происходит анализ кода, векторизация (преобразование циклов в векторные команды) и т.д.

Система команд насчитывает около 150 векторных и скалярных команд. Исполнительные устройства могут работать только с регистрами.



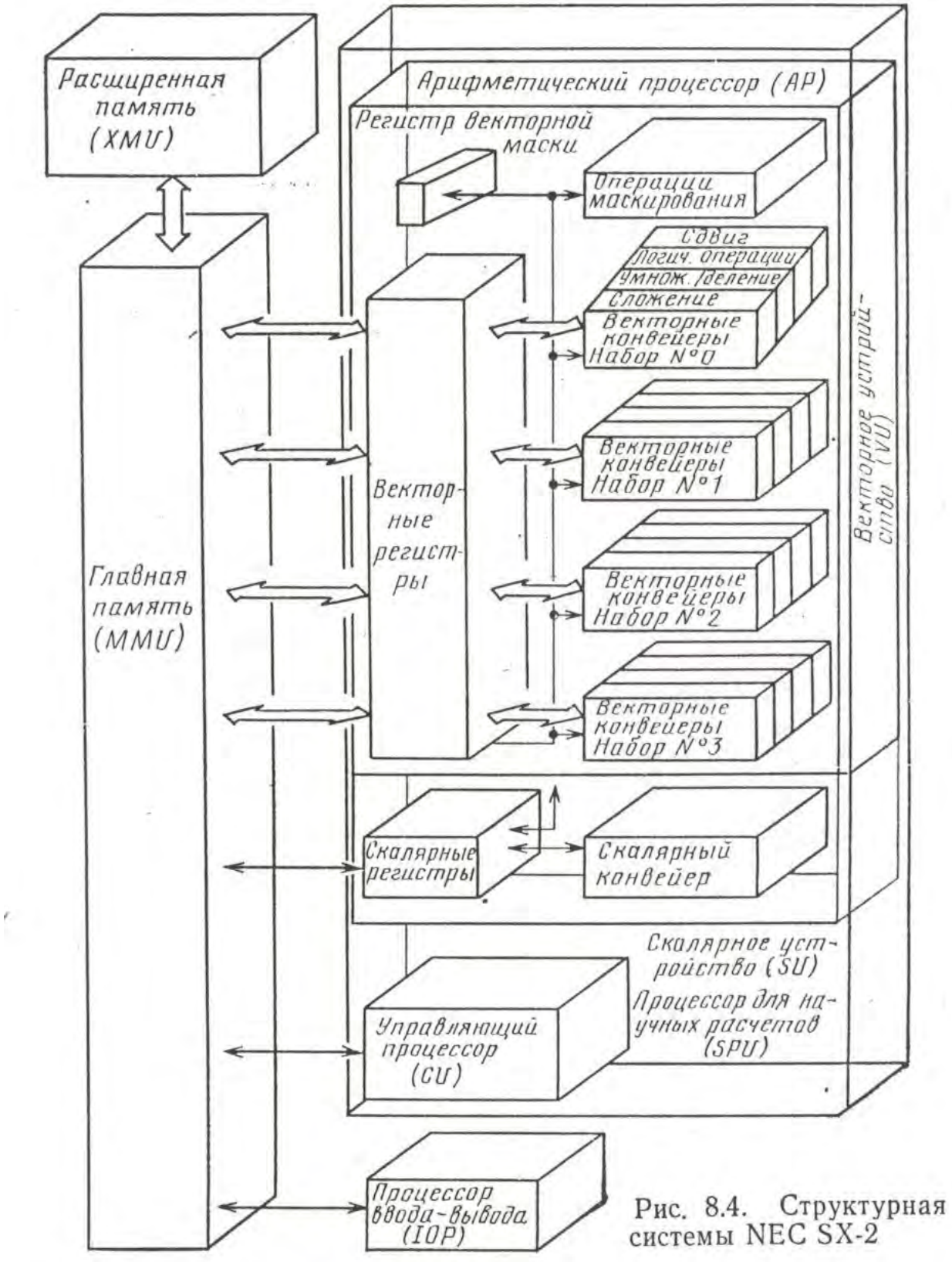


Рис. 8.4. Структурная системы NEC SX-2

Рис.18.4 Структурная схема ВС *NEC SX*-2

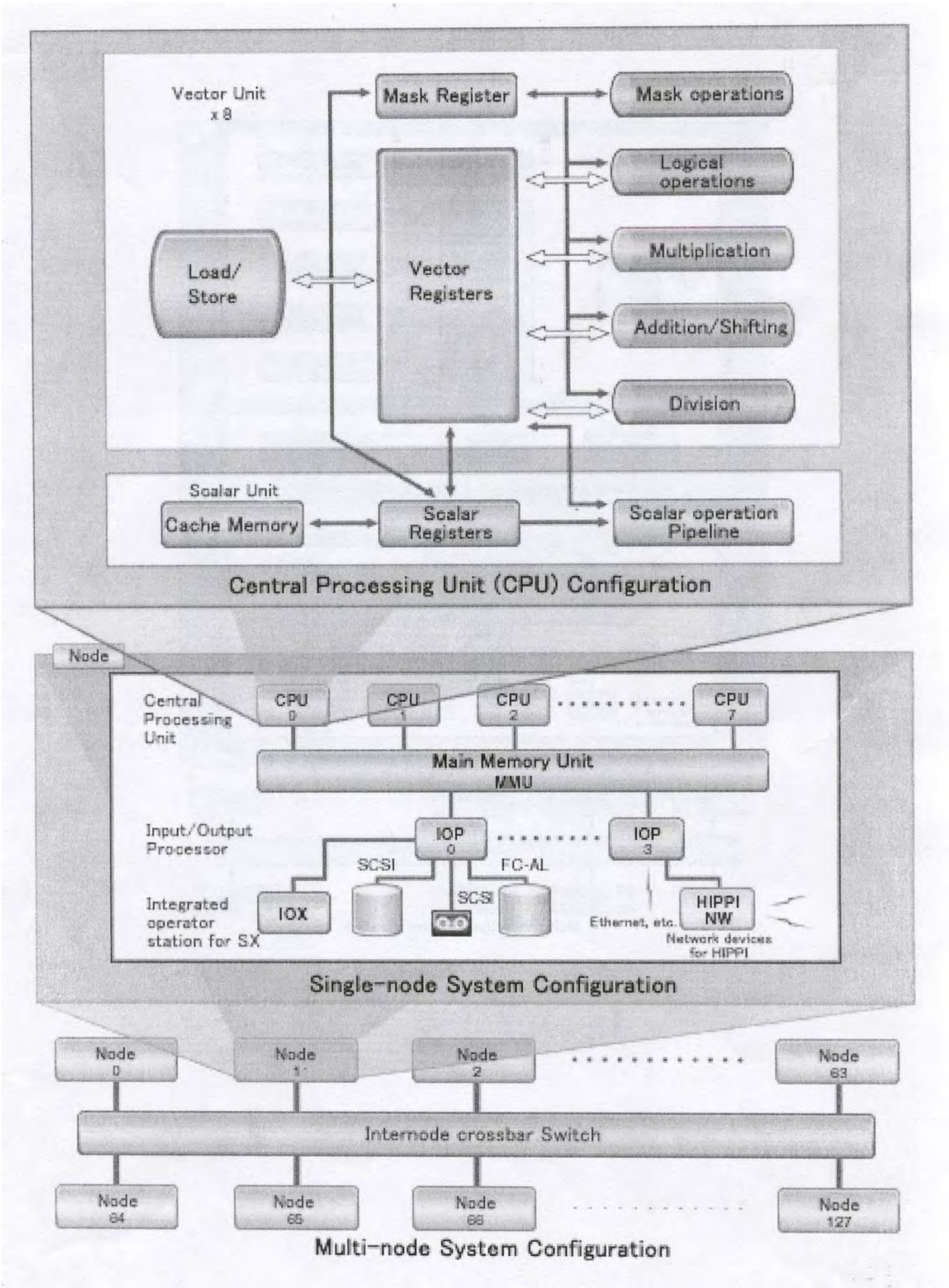
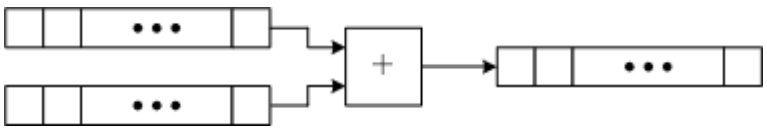
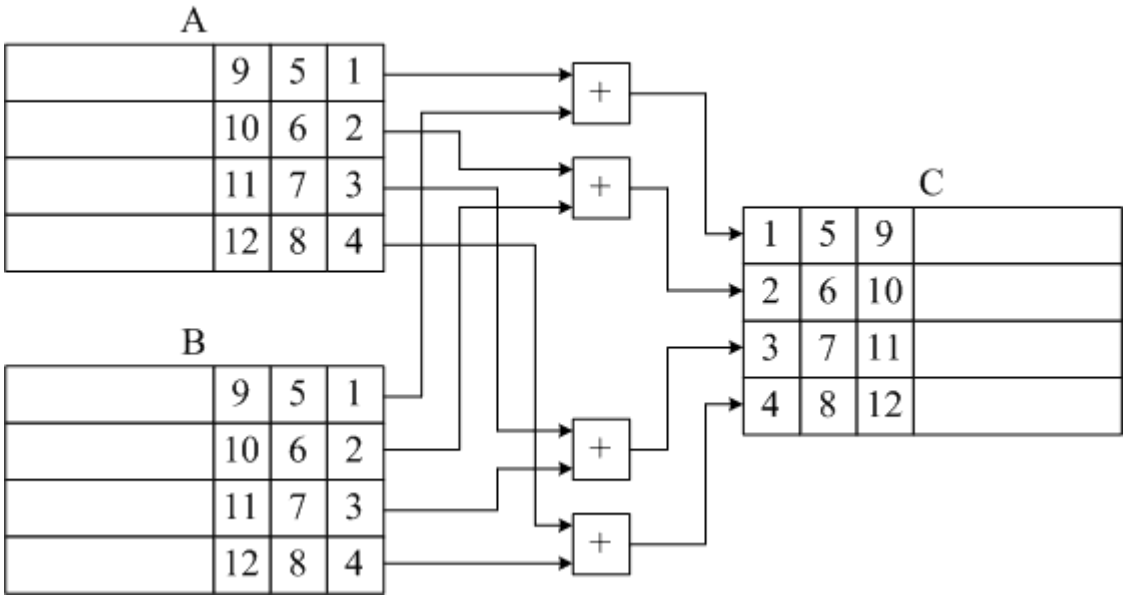


Рис.18.5 Структурная схема ВС *NEC SX*-6

Векторный процессор имеет четыре устройства по четыре одинаковых устройства; скалярный процессор имеет по одному такому устройству. Итого: 4×4 + 4 = 20 устройств.



Предусмотрено выделение четырёх векторных регистров на каждый вектор, благодаря этому операции выполняются в четыре раза быстрее.



Реализован принцип зацепления (заимствован из *CRAY*).

Память с расслоением (4 операции, следовательно, 8 операндов). Если операнды расположены подряд, тогда всё хорошо, но если имеет место смещение, то необходима операция индексации. В ВС *CRAY* это реализовано аппаратно, т.е. векторный процессор не занимается адресацией. Здесь подобных блоков нет.

Оценка производительности ВС *NEC SX*-6:
f = 1 (процессор) × 8 (операндов за раз) × 128 (узлов) ГГц!

Лекция 17

19. ВС на основе ассоциативных процессоров

19.1 Введение

Прежде чем говорить о параллельных ВС класса ОКМД, , необходимо выделить два их типа разделяемых видом обработки – числовая и нечисловая (построенных на основе ассоциативных процессоров). Как правило, для числовой обработки существенными характеристиками являются точность и время выполнения, а сами вычисления представляют собой длинные цепочки итераций. Примерами таких вычислений могут служить решение линейных и дифференциальных уравнений, преобразование Лапласа, операции с матрицами, векторами и др.

В отличие от числовой обработки нечисловая обработка связана с созданием систем обработки экономической информации, информационных систем различных применений, автоматизации управленческих работ в различных учреждениях и т. д. Следовательно, основной характеристикой систем нечисловой обработки является способность хранить огромное число различных сведений. Поэтому главным их элементом являются базы данных, при этом главными операциями становятся такие как поиск и сортировка данных.

Следует отметить, что при числовой и нечисловой обработки в понятие «данные» вкладывается различное содержание. При числовой обработке используются такие объекты, как переменные, векторы, матрицы, многомерные массивы, константы и др. При нечисловой обработке объектами могут быть файлы, записи, поля, иерархии, сети, отношения и т.д.

Любые данные в той или иной степени обладают некоторым содержанием. При числовой обработке мы хотим, например, вычислить сумму элементов массива или перемножить две переменные. При этом содержание элементов массива мало о чем говорит. Для суммирования достаточно указать адрес начального элемента массива, а затем накапливать сумму элементов в некотором регистре, индексируя соответствующим образом адреса. Даже в условном операторе мы обращаемся к элементу данных не по содержанию, а по имени. Таким образом, при числовой обработке содержание данных не имеет большого значения.

При нечисловой обработке, наоборот, нас интересуют непосредственные сведения об объектах (информация о конкретных служащих, а не файл служащих как таковой). Естественно, поэтому в системах, ведущих нечисловую обработку данных, основным компонентом становится ассоциативный процессор.

Как было отмечено выше, при нечисловой обработке данных главными операциями являются хранение, поиск и сортировка данных. Машины, специально предназначенные для реализации таких операций, получили название *машины базы данных* [7].

Исторически, первоначально, почти все базы данных содержали простые фактические данные, такие, как, например, суммы вкладов или численность населения, а объём данных был незначителен, однако эффективность использования баз данных привёло к тому, что область их применения стремительно росла, и стали появляться сложные информационные системы, которые не ограничиваются простыми фактическими данными, выполняют такие законы и правила, которыми оперируют, например, экспертные системы. В таких системах данные, называемые знаниями, могут не только накапливаться, но и при определённой обработке видоизменяться в новые знания. Таким образом, происходит переход от машин баз данных к *машинам баз знаний*. При обработке интеллектуальной информации требуется чрезвычайно высокая гибкость системы для возможности накопления знаний, имеющих сложную структуру. Современные ЭВМ, обладая высокой производительностью и многоуровневой памятью большой ёмкости, не соответствуют требованиям эффективно накапливать большие объёмы данных, производить быструю выборку требуемых данных, осуществлять поиск и специальные операции над данными. Поэтому существует необходимость создания специализированных вычислительных систем, удовлетворяющих данным требованиям.

При выборе структуры ВС и способа обработки данных (напомню, что речь идёт о нечисловой обработке данных), необходимо учитывать структуру реальных данных, между которыми существуют самые разные взаимосвязи. Формализация структуры данных, т. е представление данных в виде некоторой модели, обычно ориентируется на три основных модели данных - иерархическая, сетевая и реляционная.

Иерархическая модель представляет зависимость между записями в виде древовидной структуры. Например, данные, содержащие сведения о предприятии, затем данные о подразделениях и т. д.

Сетевая модель описывает более обобщённые связи между записями и не ограничивается представлением древовидной структурой, как иерархическая модель. Однако, несмотря на то, что она является более универсальной, более гибкой, тем не менее, она отражает узкий класс структур данных.

Реляционная модель описывает структуры данных на основе комбинаций математического множества и отношений между n членами и представляет данные в виде таблицы. Обычно такую таблицу называют отношением:

название дисциплины	название лекции	лектор
организация ЭВМ и ВС	ЗУ	Дерюгин
вычислительная система	матричные системы	Ладыгин
поисковое проектирование	курс – 2	Дзегелёнок
основы теории ВС	автоматы	Дзегелёнок

Поскольку в реляционной модели почти не накладываются ограничения на физическую структуру, представление данных с помощью этой модели оказывается чрезвычайно гибкой.

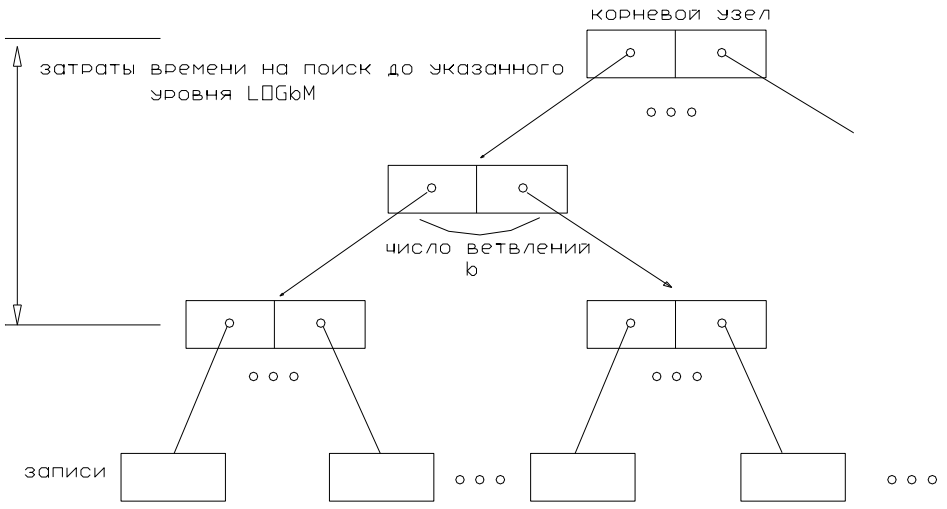
Таким образом, ориентируясь на ту или иную модель данных, можно выбрать соответствующий способ организации обработки данных и, следовательно, структуры ВС.

Как было отмечено выше, одной из основных операций машин баз данных является поиск данных, удовлетворяющих определённым условиям. В общем случае, если имеется m записей, для поиска записи, удовлетворяющей некоторым условиям, могут быть использованы следующие три основных метода:

1. ассоциативный поиск;
2. поиск по дереву;
3. последовательный поиск.

Первый метод предполагает использование ассоциативной памяти и методов хэширования. При этом затраты времени на поиск обычно постоянны и не зависят от значения m.

В соответствии со вторым методом строится древовидная структура, в которой узлы связаны указателями. Во все записи, расположенные в углах, занимающих самые нижние позиции, вводятся указатели записей. Поиск начинается с верхних (так называемых корневых) узлов и продолжается от узла к узлу, пока не будет достигнута искомая запись.



По пути в каждом узле осуществляется сравнение с ключом поиска, и дальнейший маршрут поиска определяется в зависимости от результата сравнения (больше – меньше). Таким образом, в каждом узле должна быть помещена информация о верхнем (нижнем) пределе значения ключа. Затраты времени зависят от высоты дерева и числа сравнений.

Третий метод поиска наиболее простой: m записей последовательно сравниваются с ключом поиска. Поскольку считается, что расположение искомой записи подчиняется равновероятному закону, средние затраты времени на поиск определяются как m/2.

Оценка затрат времени на поиск, приведённая выше, относится к случаю, когда все записи размещены в оперативной памяти, обеспечивающей одинаковое время доступа к записям. На практике оказывается, что в оперативной памяти находится только небольшая часть данных, а остальные располагаются в дисковой памяти, тогда проблема определения времени поиска становится сложной.

Сокращение количества данных, передаваемых между оперативной памятью и дисковой и уменьшение времени поиска данных возможно с помощью реализации так называемой «системы интеллектуальных дисков», в которой функции поиска переданы устройству управления дисками, и набора логических ячеек, составляющих логическую схему вместе с блоком магнитных головок, в котором каждой дорожке соответствует своя головка.

В машинах баз данных, ориентированных на использование реляционной модели данных, кроме операции поиска данных, основными операциями являются также и операции, необходимые для обработки данных. В рассматриваемом случае необходима такая структура ВС, в которой в качестве языка манипулирования данными для реляционных баз данных аппаратно реализуется реляционная алгебра, т. е это операции над множествами: суммирование, вычитание, пересечение, получение декартова (прямого) произведения, и операции над отношениями: выборка, проецирование, соединение, деление и т.д. Наиболее важными операциями над отношениями являются выборка, соединение и сортировка. Если в машине реализованы эти три функции, остальные операции могут выполняться просто и эффективно.

19.2 Ассоциативные процессоры

Ассоциативным процессором называют ассоциативное запоминающее устройство (АЗУ), дополненное логикой и микропрограммным управлением. Как известно, логика имеется и в самом АЗУ, поэтому иногда АЗУ также называют ассоциативным процессором. Однако последний обычно обладает большими возможностями обработки данных, чем простой поиск. Параллельная обработка при выполнении одной и той же операции (например, операции сравнения) присуща самому АЗУ. Поэтому ассоциативные процессоры относят к классу SIMD – архитектур. Ассоциативным процессорам посвящено значительное число исследовательских работ. Большие ассоциативные процессоры из – за высокой стоимости не получили широкого развития (за редким исключением). АЗУ небольшой емкости используются в современных фоннеймановских машинах. АЗУ применяются в тех случаях, когда требуется быстрый просмотр. Их можно использовать в кэш – памяти, для

поиска страниц в системах виртуальной памяти или в любых других подсистемах, требующих быстрого поиска по таблице.

Существует классификация ассоциативных посессоров по уровню распределенности аппаратной поддержки (логики). Логика может быть распределена до уровня бита, байта, слова или более крупного блока памяти. Чем ниже этот уровень, тем выше уровень параллелизма (и соответственно – стоимость).

Ассоциативный процессор обладает (помимо способности к запоминанию информации) некоторой «интеллектуальностью». Этой интеллектуальности вполне достаточно для выполнения следующих операций:

- совпадает (равно);
- не совпадает (не равно);
- меньше;
- меньше или равно;
- больше;
- больше или равно;
- максимум;
- минимум;
- в пределах;
- вне пределов;
- ближайшее большее;
- ближайшее меньшее;
- различные виды сложения и вычитания.

19.3 Ассоциативные процессоры с пословной организацией

В основе архитектуры ассоциативных процессоров с пословной организацией лежит параллелизм на уровне слов, и в большинстве конфигураций обработка слов выполняется последовательно по разрядам. Множество слов образует ассоциативный массив или АЗУ пословно организованного ассоциативного процессора (АП). Соответственно имеется по одному процессорному элементу на каждое слово, так что весь разрядный срез АЗУ может обрабатываться параллельно. Ниже мы рассмотрим особенности пословно организованных ассоциативных процессоров, а затем опишем некоторые реализации таких систем.

19.3.1 Базовая структура

Базовая структура пословно организованного АП содержит следующие подсистемы:

- АЗУ (массив ассоциативной памяти);
- регистр компаранда;

• регистр маски;
• регистры хранения ответов;
• регистры или буфер ввода – вывода АЗУ;
• маска (буфер) вывода слов;
• соединительная сеть;
• контроллер (память и программы).

Очевидно, что данная структура подобна базовой модели АЗУ, рассмотренной в предыдущем пункте. Однако, как мы покажем ниже, ассоциативный процессор имеет дополнительные функции и возможности.

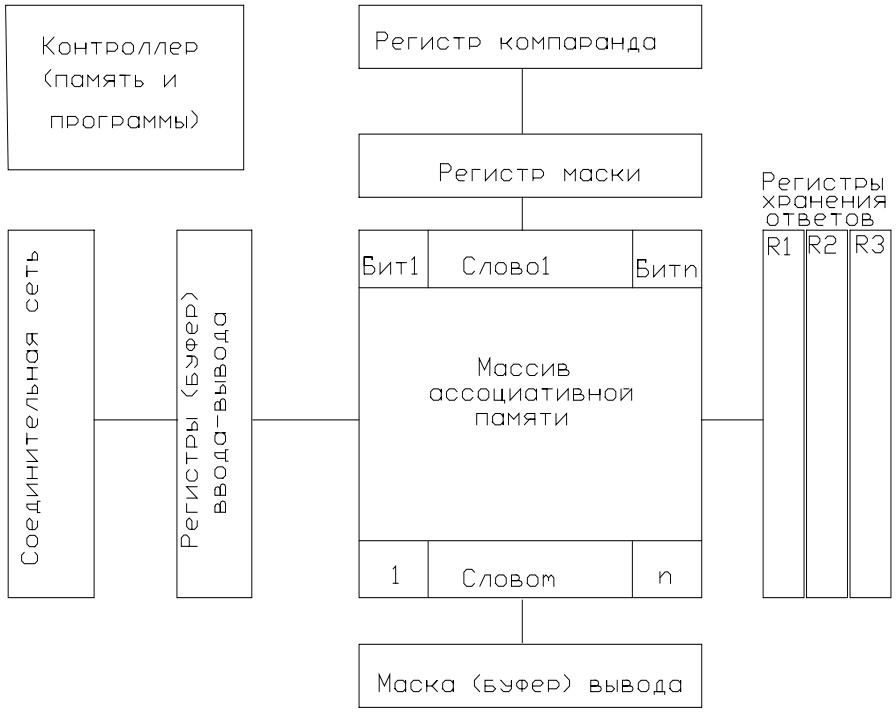


Рис. Базовая структура пословно организованного АП.

Рассмотрим подробнее перечисленные выше подсистемы.

Массив ассоциативной памяти. Этот массив содержит m n – разрядных слов. При нечисловой обработке такая запоминающая матрица размером m x n является удобной средой для отображения двумерных логических структур, таких, как матрицы, файлы и отношения. Каждая строка матрицы, каждая запись файла или каждый кортеж отношения соответствует слову ассоциативного массива. Хотя в таких структурах и допускается переполнение с переносом в следующие слова, типичная

характеристика этих систем состоит в том, что каждое слово массива предназначается для одной строки, записи или кортежа соответствующих структур. Это объясняется несколькими причинами. Одной из них являются физические ограничения реализации, связанные с длиной слов. Начиная с некоторой длины слова, в аппаратуре возникает проблема шумов. Другая причина состоит в том, что периферийная аппаратура поиска массива ассоциативной памяти может выполнять одновременно лишь один поиск, если не вводить дорогостоящего дублирующего оборудования. Исключением из этого правила является разработанная в последнее время система Lucas. В этой системе атрибуты или поля, принадлежащие более чем одному отношению или файлу, могут храниться вместе в одном слове, так что к ним можно применять бинарные операции реляционной алгебры. Поскольку, однако, ограниченная длина слов массива может не вмещать полных записей, это может потребовать дополнительного времени для формирования кортежей или записей. Может потребоваться также дополнительный ввод – вывод из внешней памяти для заполнения оставшихся частей кортежей или записей, предназначаемых для поисковых операций или обработки баз данных.

Другая характеристика этих процессоров состоит в том, что поскольку операции выполняются в них пословно и поразрядно, они требуют высокой скорости чтения и записи. Это в свою очередь обусловливает выбор в качестве памяти статических запоминающих устройств вместо более медленных, хотя и более емких динамических. С этим типом памяти совместимы слова длиной 256, 1024 и даже 4096 бит (как мы увидим в следующих главах, одна ячейка машины баз данных RAP.3 содержит основную память емкостью 2 Мбайт; ячейка в данном случае соответствует отдельному слову АП). При горизонтальных размерах такого порядка требуется вертикальный размер (или параллелизм, по словам), соответствующий кардинальному числу (т.е. числу записей файла или кортежей отношения) хранимой в них структуры. Чтобы выполнить соединение двух отношений, каждое из которых содержит 100 000 кортежей, потребовалось бы АП, имеющий АЗУ емкостью 200 000 слов (или 100 000 в системе Lucas). Но до сих пор ни разу не строились и даже не предлагались машины такого размера!

Регистр компаранда. В массиве АЗУ могут выполняться различные комбинации операций поиска. Это означает, что мы можем сравнивать некоторое внешнее значение со всеми словами, хранимыми в АЗУ; мы можем сравнивать друг с другом два поля во всех словах; наконец, мы можем сравнивать некоторое поле выбранного слова с полями других слов. Но во всех этих случаях компаранд должен быть сначала введен в регистр компаранда, чтобы выполнить массовое сравнение (или сравнение типа SIMD), в котором одно значение сравнивается параллельно со множеством значений. При параллельном сравнении двух полей во всех словах имеет место параллелизм другого типа, и самый эффективный способ этого

сравнения состоит в выполнении логической операции или операции вычитания между двумя полями в каждом слове, так что конечный результат является искомым результатом сравнения. Таким образом, хотя данные, т.е. компаранды, в каждом слове различны, операция выполняется параллельно по словам и последовательно по разрядным срезам. Длина регистра компаранда соответствует длине слова АЗУ. В типичном режиме сравнения позиции разрядов компаранда, маски и слов АЗУ соответствуют друг другу. В общем случае в таком соответствии нет необходимости, поскольку можно направить некоторое поле компаранда для сравнения с полем, расположенным в другой области АЗУ.

Регистр маски. Регистр маски можно рассматривать как простой фильтр или селектор разрядов регистра компаранда. С помощью регистра маски указывается поле регистра компаранда, которое служит действительным компарандом. Все разряды, не входящие в указанный компаранд, маскируются (т.е. соответствующие разряды регистра маски устанавливаются в 0), с то время как разряды, соответствующие полю компаранда, устанавливаются в 1.

Регистры хранения ответов. Обычно каждый регистр хранения ответов представляет собой множество триггеров, образующее одноразрядный двоичный вектор вдоль всего массива АЗУ. Один из этих регистров называется *регистром меток (тегов)* или *отклика* и служит для идентификации и хранения результатов операций над массивом. Выделенное слово АЗУ, т.е. слово, отмеченное единицей в регистре меток, обычно индицирует успешное окончание поиска или хранит результат логической операции (например, перенос в текущем шаге операции сложения). Второй регистр памяти ответов используется как временная рабочая область (или сверхоперативная память) при обработке данных или реализации логических функций. Третий регистр памяти ответов служит для выбора слов. Он указывает для каждого слова АЗУ, участвует ли оно в подлежащей выполнению операции. Если, например, в массиве АЗУ хранятся файлы А и В и мы хотим выполнить некоторую операцию над одним из этих файлов, то необходимо запретить обработку всех слов другого файла путем установки в 0 всех разрядов регистра выбора слов, соответствующих второму файлу.

Регистры (или буфер) ввода – вывода АЗУ. Эти регистры играют роль буфера при передаче данных в АЗУ или из него. Манипулирование данными может относиться к одному слову или распространяться на весь массив. Последнее происходит обычно при загрузке и разгрузке АЗУ. В этом случае самое важное – выполнить эту операцию как можно быстрее. В самом деле, поскольку объем АЗУ относительно небольшой по сравнению с объемом подлежащих обработке данных, такие операции потребуется выполнять очень часто, при этом массив АЗУ не должен долго бездействовать в ожидании ввода и вывода данных. Поэтому были использованы специальные интерфейсы ввода – вывода, такие, как диски с фиксированными головками и барабаны, в которых каждая головка (т.е.

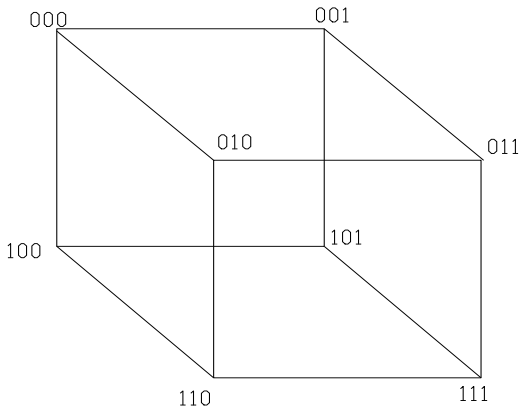
тракт) предназначалась для одного слова АЗУ. Если массив АЗУ состоит из 256 слов, такой интерфейс обеспечивает параллельный канал шириной 256 бит, имеющий полосу пропускания в 256/8=32 раза большую, чем полоса стандартного последовательного по байтам канала ввода – вывода.

Маска (буфер) вывода слов. Этот буфер используется для последовательного по словам вывода содержимого АЗУ. Возможность маскирования позволяет выбирать отдельные поля по тому же принципу, что и в регистре маски.

Соединительная сеть. В рассматриваемом классе ассоциативных процессоров соединительная сеть используется для логических комбинаций полей в данной физической среде, чтобы можно было выполнить требуемые операции. Иначе говоря, соединительная сеть эмулирует на двумерном массиве АЗУ более сложные виды межпроцессорных соединений. Например, при обработке изображений или геометрических фигур возможна работа с сеточными структурами, где необходимо соединение каждого элемента с четырьмя ближайшими соседями. Может потребоваться также сеть «идеальной тасовки» для сортировки, вычисления полиномов и т.д. В случае идеальной тасовки, если процессоры нумеруются от p_0 до p_{n-1} и позиция процессора соответствует двоичному числу $p_{n-1}\, p_{n-2} \ldots p_2\, p_1\, p_0$, процессор будет соединен с процессором $p_{n-2} \ldots p_2\, p_1\, p_0\, p_{n-1}$. Иными словами, тасовка эквивалентна циклическому сдвигу на единицу двоичной записи номера процессора. Последовательности из восьми процессоров (01234567) после однократной тасовки соответствует последовательность (02461357); это означает, например, что процессоры 2 и 4 соединяются соответственно с процессорами 4 и 1 (поскольку 010 и 100 превращаются после сдвига в 100 и 001).

Схема кубического соединения, которая для трехмерного случая приведена на рисунке ниже, может применяться для обработки трехмерных изображений.

Например, для сглаживания изображения значение интенсивности в точке (i,j) может быть заменено средним значением в этой же точке и в восьми соседних, т. е. (i-1,j), (i-1,j-1), (i,j-1), (i+1,j), (i+1,j+1), (i,j+1), (i-1,j+1) и



(i+1,j-1). Для растра 2^k X 2^k необходимо вычислить 2^{2k} средних значений, для чего понадобится столько же итераций усреднения при использовании одного процессора. Если, однако, у нас имеется АП (ассоциативный процессор), состоящий из $m=2^n$ процессоров, изображение можно разделить на $\sqrt{m}X\sqrt{m}$ фрагментов размером $2^k/\sqrt{m}$ X $2^k/\sqrt{m}$. Все эти m фрагментов можно сглаживать параллельно, уменьшив тем самым число итераций при вычислении средних значений до $2^{2k}/m$.

Контроллер. Массив АЗУ и все рассмотренные выше схемы управляются контроллером. Запрос пользователя транслируется в основной машине в базовые операции АП и посылается в котроллер. Котроллер имеет память для программ, реализующих базовые операции, и различные регистры, логику управления шиной, механизмы прерываний и арифметико-логическое устройство для вычисления адресов и сдвига информации. Каждая конкретная система имеет собственную конструкцию контроллера.

19.3.2 Базовые операции

Как мы видели выше, разрядный срез является основной конструкцией в пословно организованных АП. Существуют операции считывания и записи разрядного среза, пересылки содержимого одного разрядного среза в другую позицию, логической обработки разрядных срезов и т.д.

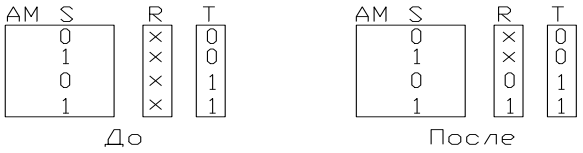
Загрузить разрядный срез (Load bit slice). Эта операция пересылает содержимое некоторого разрядного среза АЗУ в один из регистров хранения ответов. Введём следующие обозначения: R – триггер регистра выбора слов в памяти ответов; T – триггер регистра меток в памяти ответов; S – позиция источника; D – позиция назначения.

Теперь приведём три варианта операции «Загрузить разрядный срез»:

а) LOAD R /* загрузить разрядный срез в вектор R /

б) LOAD R [T] /* загрузить в R отмеченные метками T биты разрядного среза */

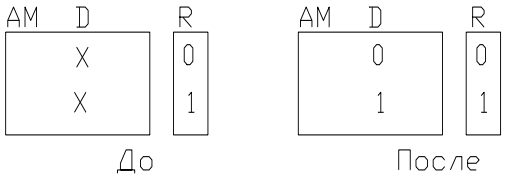
в) LOAD T [T] /* загрузить в T биты разрядного среза, отмеченные прежними значениями меток T */



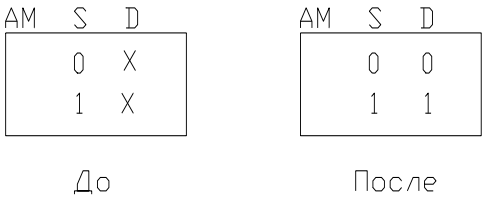
Операция LOAD R [T].

Запомнить разрядный срез (Store bit slice). Эта операция противоположна операции загрузки. На рисунке ниже показан пример выполнения операции STORE R. Как в случае загрузки, так и в случае запоминания разрядного среза позиции источника (S) и назначения (D) указываются контроллером (точнее, адресным процессором контроллера).

Переслать разрядный срез (Move bit slice). Данная операция пересылает содержимое одного разрядного среза в позицию другого. При этом используются триггеры памяти ответов, т.е. сначала выполняется операция «загрузить», а затем – «запомнить». На рисунке ниже показано действие операции MOVE BITSLICE.



Операция STORE R.

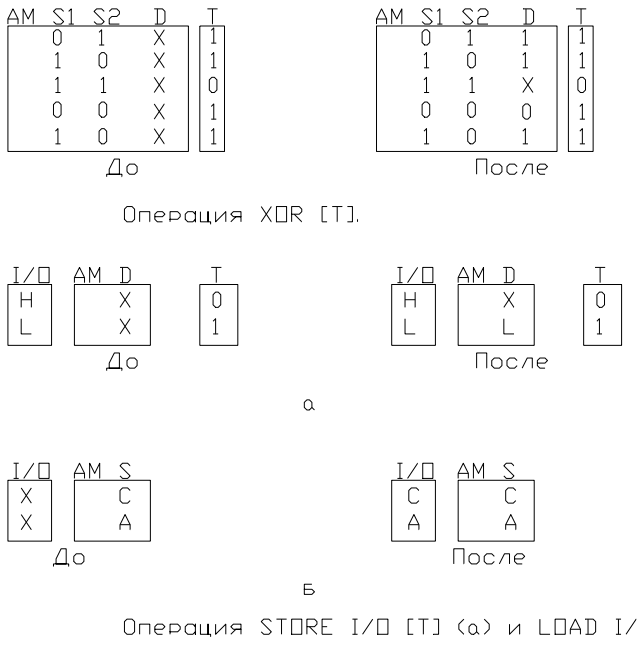


Операция MOVE BITSLICE.

Логические операции над разрядными срезами. В этом случае два разрядных среза подвергаются логической операции, и результат помещается в позицию (адрес) некоторого третьего разрядного среза. Здесь также используются триггеры памяти ответов. Для любой операции первый из операндов загружается в какой-либо триггерный вектор (например, R), заданная логическая операция выполняется между вторым операндом (т.е. соответствующим разрядным срезом) и триггерным вектором R, после чего результат, заменивший прежнее содержимое триггерного вектора, пересылается из этого вектора в позицию результирующего разрядного среза. Логической операцией может быть любая булева функция двух переменных. На рис. ниже представлены 2 из 4-х этапов выполнения операции XOR[T]

Выбрать первого ответчика и сбросить (Select first and reset). Обычно в архитектурах рассматриваемого типа для любой операции имеются несколько ответчиков, так как операция вырабатывает множество результатов. Однако при некоторых операциях последовательного характера, например при работе с указателями, в данный момент времени должен обрабатываться только один из результатов. Затем метка первого результата удаляется из вектора меток, производится переход к следующему результату и т.д. Соответствующие шаги этой операции выполняются командами SELEKT FIRST (или GET FIRST) и RESET. В векторе меток выделяется первое слово, отмеченное единицей, запоминается его адрес, после чего данная метка сбрасывается.

Операции с байтами. Операции с байтами аналогичны операциям с битами и выполняются путём итерации последних. В операциях с байтами также осуществляется ввод-вывод через регистр ввода-вывода и внешний общий регистр. На рисунке ниже показаны операции STORE I/O [T] и LOAD I/O в предположении, что каждому слову АЗУ соответствует однобайтовый регистр.



Обычно устройство управления имеет общий регистр, который может обмениваться данными с регистрами ввода-вывода массива АЗУ. Поскольку имеется лишь один общий регистр, операция его загрузки из массива предполагает выбор первого ответчика и последующий сброс.

На рисунке ниже показана операция LOAD COM между массивом и общим регистром (COM). При выполнении аналогичной операции в обратном направлении содержимое общего регистра дублируется во всех регистрах ввода-вывода.

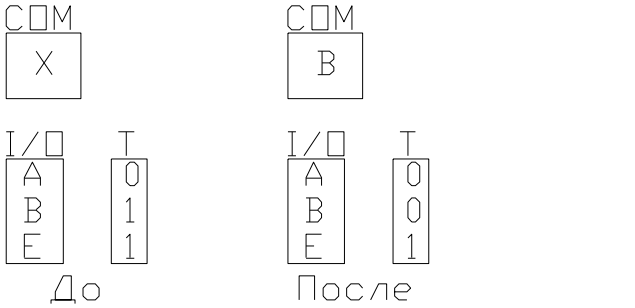
Имеются также операции пересылки данных между регистрами компаранда и маски и соответствующими буферами ввода-вывода.

Аналогично операциям над разрядными срезами байтовый срез может быть перемещён на место своего назначения, как это делается, например, в операции MOVE BYTESLICE [T]. Содержимое регистра-компаранда также может быть передано в массив АЗУ. На рисунке ниже показана операция STORE CBYTE [T].

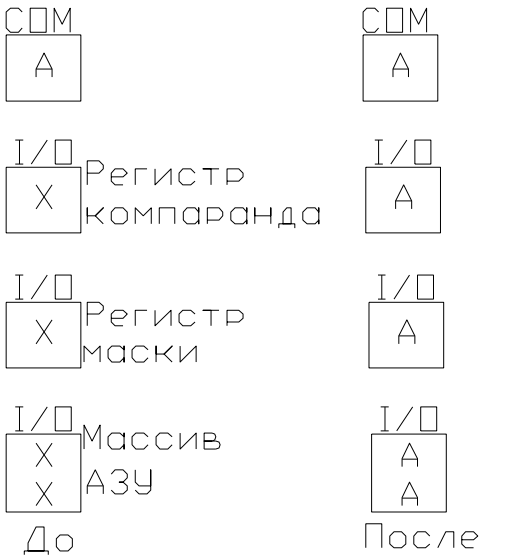
Операция дублирования может быть также использована для пересылки некоторого поля данного слова АЗУ во все остальные слова. На рисунке ниже показана операция SPREAD BYTE. Как можно видеть, выполнение этой операции сводится к операциям, определённым выше (т.е. LOAD I/O, LOAD COM, SPREAD COM и STORE I/O; при выполнении операции LOAD COM существенно наличие меток).

Предусмотрен также обратный путь пересылки данных из массива АЗУ в регистр компаранда. На рисунке ниже показана операция LOAD CBYTE.

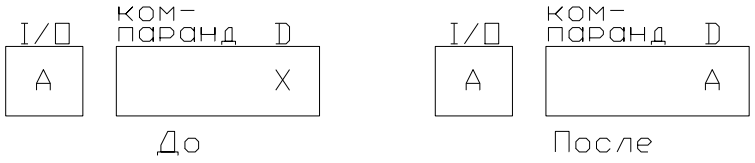
Операции с полями. Подобно пересылке байтовых срезов, могут пересылаться также поля внутри слов АЗУ. На рисунке показана операция MOVE FIELD, в которой один разрядный срез массива АЗУ используется в качестве вектора меток для поля источника, а другой – для поля назначения (соответственно разрядные срезы SM и DM). Подобно операциям над байтами, поля могут пересылаться между регистром компаранда и массивом АЗУ.



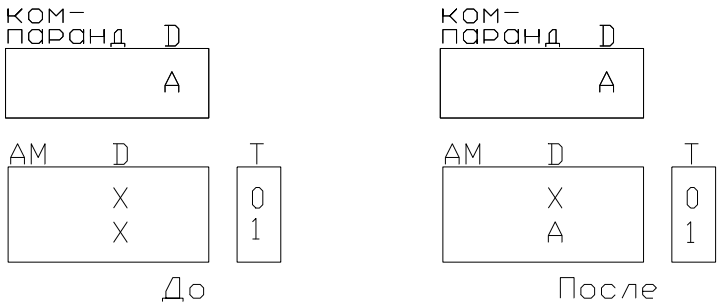
Операция LOAD COM.



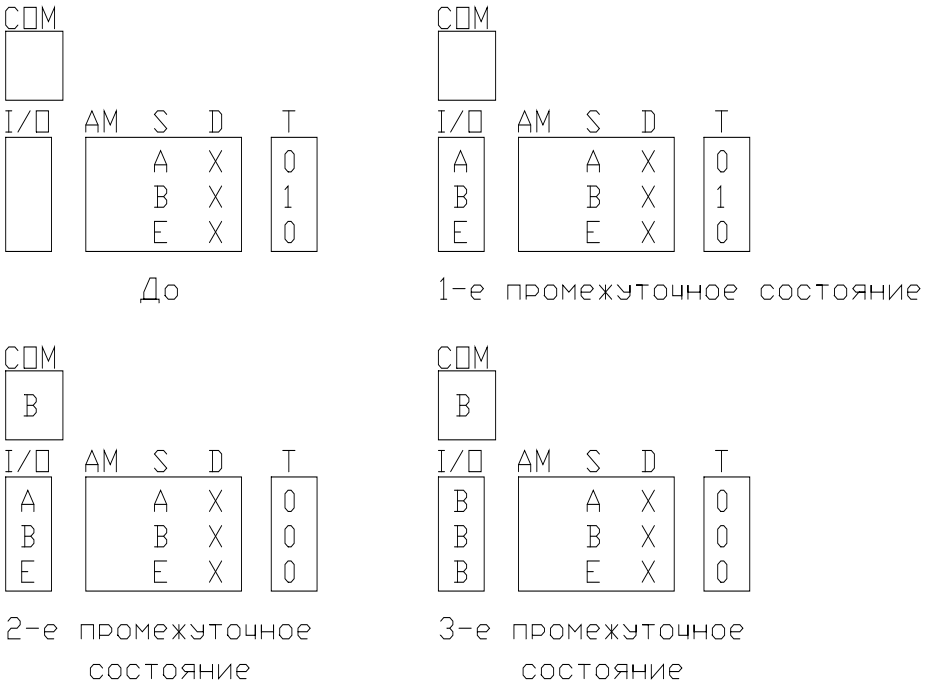
Операция SPREAD COM.

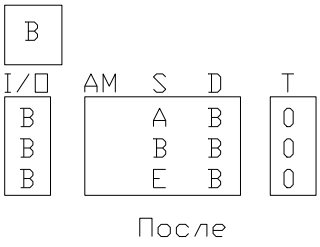


Операция LOAD COMPARAND.

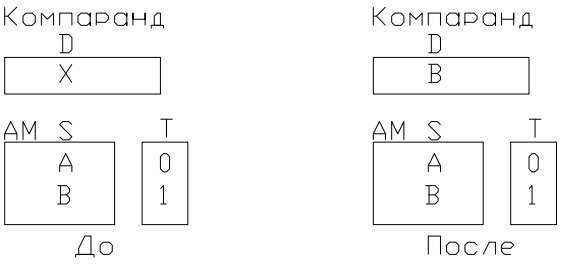


Операция STORE CBYTE [T].

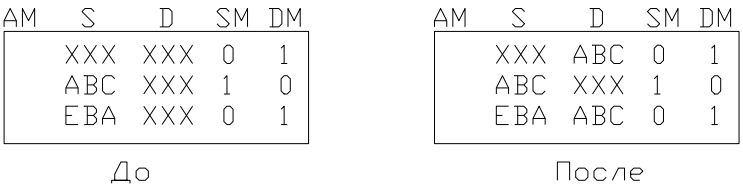




Операция SPREAD BYTE.



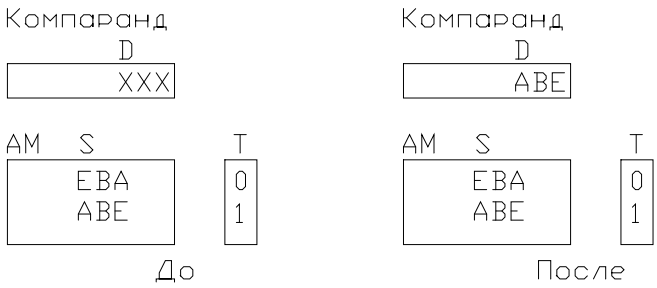
Операция LOAD CBYTE.



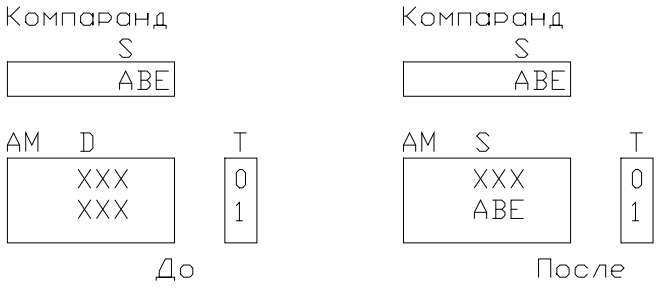
Операция MOVE FIELD.

На рисунке ниже показаны операции LOAD CFIELD и STORE CFIELD.

При выполнении операций поиска в массиве АЗУ система работает как стандартное ассоциативное запоминающее устройство. Поле массива АЗУ, подлежащее сравнению, может быть выделено регистром маски, либо адресный процессор контроллера может генерировать последовательность адресов разрядных срезов соответствующего поля. Результаты сравнения отмечаются в векторе меток T.



а



б

Операция LOAD CFIELD (а) и STORE CFIELD (б).

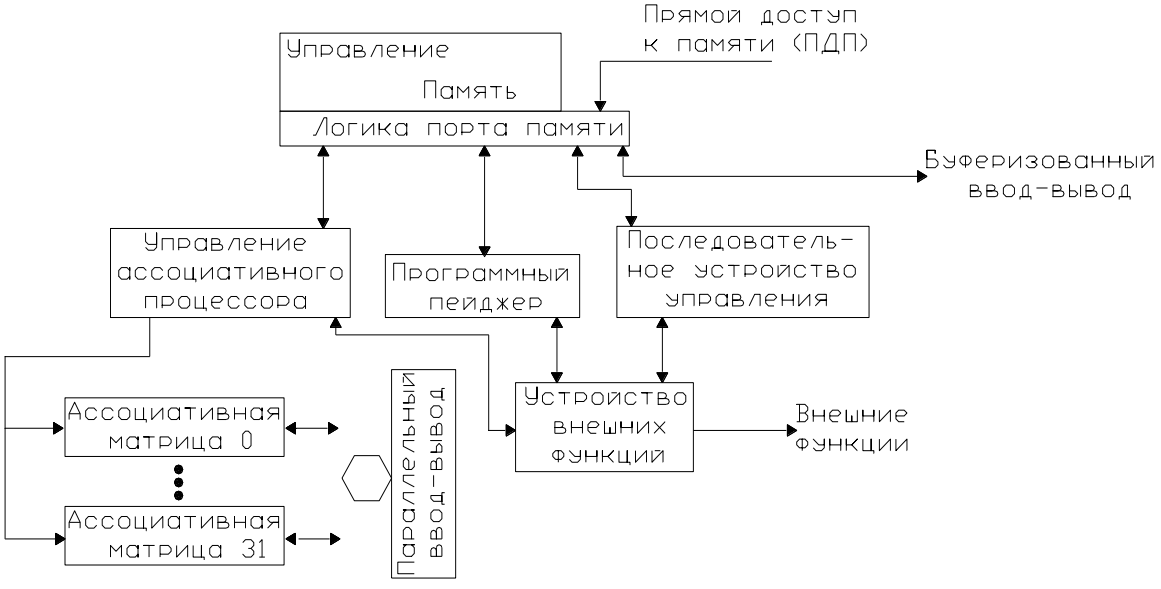
19.4 Ассоциативный процессор Staran

Ассоциативный процессор Staran фирмы Goodyear Aerospace является единственным ассоциативным процессором, который изготовляется серийно и продаётся. В течение ряда лет он служил стандартным примером ассоциативного процессора, приводимым в учебниках. В процессе развития он прошёл путь от модели Staran-B, имевшей до 32 блоков (ассоциативных модулей) размером 256x256 бит, до модели Staran-E, каждый блок которой имеет 256 слов по 9216 бит.

Процессор Staran был первоначально построен для обработки радиолокационных сигналов в военных применениях, но его ассоциативная параллельная структура хорошо приспособлена и для различных задач нечисловой обработки, например для управления базами данных.

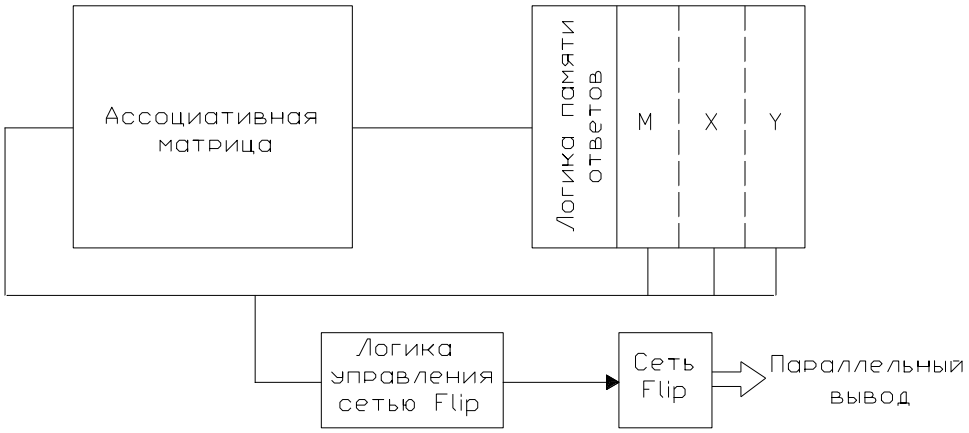
На рисунке ниже показана общая организация системы Staran, а на следующем рисунке – организация одного ассоциативного модуля системы. В системе Staran выполнимы все типичные и характерные операции над массивом АЗУ, описанные в начале этой главы. Для установки меток используется вектор Y (т.е. триггеры Y_i для каждого I-го слова АЗУ). M_i есть разряд выборки слова, а X_i используется в качестве временной (сверхоперативной) памяти. В обозначениях системы Staran F_i представляет I-й разряд разрядного среза, а также используется для описания перестановок соединительной сети Flip.

Управляющая память ассоциативного процессора Staran организована послойно для размещения различных уровней памяти и блока замещения страниц (пейджера). Этот блок распределяет страницы между различными уровнями управляющей памяти, не задерживая выполнение операций в массиве АЗУ. На верхнем уровне управляющей памяти находятся три страницы быстрой полупроводниковой биполярной памяти, которые содержат программы, управляющие операциями в массиве АЗУ. Ниже по



Структура системы Staran

уровню располагается быстродействующий буфер ёмкостью в одну страницу, работающий при пересылке страниц между биполярной памятью и относительно медленной ферритовой памятью. Каждая страница содержит 512 слов, в то время как основная ферритовая память имеет ёмкость 16К слов. Вся управляющая память ассоциативного процессора соединена с внешними элементами через специальный порт памяти, работающий с учётом приоритетов. В адресном пространстве иерархии управляющей памяти ассоциативного процессора зарезервирована область между находящимися на высшем уровне быстродействующим буфером и биполярным ЗУ и находящейся на нижнем уровне ферритовой памятью. К этим зарезервированным областям памяти может быть добавлена, в случае необходимости, схема прямого доступа к памяти (ПДП) для доступа к памяти основной машины, что обеспечивает прямую связь основной машины с системой Staran. Внешние связи быстродействующего буфера и ферритовой памяти осуществляются через буферизованный канал ввода-вывода.



Модуль ассоциативной памяти системы Staran.

Все подсистемы системы Staran соединены с блоком внешних функций, который служит для координации синхронизации. В этом устройстве имеется схема упорядоченного опроса, проверяющая статус всех подсистем. Все запросы, поступающие в эту схему, обслуживаются по принципу FCFS (first-come, first-served – первым пришёл, первым обслужен). Различные компоненты системы соединены друг с другом через блок внешних функций, который организует также связи страниц управляющей памяти ассоциативного процессора с соответствующими устройствами системы. Для тестирования и программирования некоторые установки блокировочных триггеров зарезервированы для внешнего использования. Например, состояние пейджера может быть проверено и при необходимости изменено. С помощью этого же блока может быть обнаружено прерывание и обеспечено взаимодействие некоторых подсистем с последовательным устройством управления или с основной машиной.

Управляющее устройство ассоциативного процессора содержит необходимую логику и регистры для управления операциями в массиве АЗУ. В состав управляющего устройства ассоциативного процессора входят обычный счётчик команд, регистры команд, общий регистр, память ответов и схемы управления массивом, указатели полей, счётчики разрядности, а также схемы сдвигов и управления шинами. Вычисления адресов массива АЗУ могут быть выполнены в этом устройстве различными способами.

Процессор Staran имеет богатые возможности манипулирования на уровне разрядов при адресации, организации внешних связей, логических и арифметических операций, а также различных операций в памяти ответов. Хотя архитектура этой системы и не похожа на архитектуру машин баз данных, она используется в самых разнообразных целях и имеет мощность, достаточную для любых применений. Ниже перечислены некоторые команды массива АЗУ.

- Загрузить память ответов из общего регистра. Эта операция имеет различные модификации, при которых выполняются сдвиги, отражение (т.е. обращение порядка разрядов) и логические операции между общим регистром и различными регистрами памяти ответов (например, Y,X).
- Загрузить память ответов непосредственно из регистра (операция регистр – регистр в памяти ответов).
- Загрузить память ответов непосредственно из массива АЗУ.
- Загрузить память ответов из массива АЗУ косвенным способом: используя различные указатели полей, можно избирательно оперировать со словами или разрядными срезами.
- Запомнить содержимое памяти ответов в массиве АЗУ.
- Те же операции с использованием регистра маски.

- Загрузить общий регистр из массива АЗУ или запомнить его содержимое.

- Найти первого ответчика.
- Сбросить первого ответчика.
- Найти и сбросить первого ответчика.
- Сбросить остальных ответчиков.

При использовании сети Flip управление отдельными каскадами этой сети осуществляется с помощью n-разрядного вектора $F=f_{n-1},\ldots, f_1, f_0$, задающего способ соединений между каскадами сети. Если $f_i=1$, то I-й каскад выполняет свою перестановку, а при $f_i=0$ осуществляется прямая связь между одноимёнными разрядами соседних каскадов. Так , для организации связей, соответствующих топологии куба при числе процессоров N=8, требуется использовать три каскада ($8=2^3$). В каждом из этих трёх каскадов имеются четыре переключательные ячейки 2X2, получающие восемь входов (по два на ячейку) и вырабатывающие восемь выходов. Передача выходов каждого каскада на входы следующего выполняется в соответствии со значением F, как было сказано выше.

Операции в массиве. Каждый процессор, соответствующий одному слову системы Staran, содержит три разряда M_i, X_i, Y_i соответствующих векторов. Эти разряды называются также одноразрядными регистрами. Каждый процессор может реализовать шестнадцать булевых функций двух переменных. Если X_i- состояние I-го разряда регистра X, а f_i-состояние I-го выхода сети Flip, то $X_i \leftarrow \Phi (X_i, f_i)$ (I=0, 1, …, 255), где Φ - булева функция. Аналогично для Y_i: $Y_i \leftarrow \Phi (Y_i, f_i)$ (I=0, 1, …, 255).

Программист может работать отдельно над X, отдельно над Y или над обоими регистрами вместе. Если работа выполняется совместно над X и Y, то к обоим регистрам применяется одна и та же булева функция Φ:

$X_i \leftarrow \Phi (X_i, f_i)$, $Y_i \leftarrow \Phi (Y_i, f_i)$.

Лекция 18

20. Особенности программного обеспечения параллельных систем

Широкое применение многопроцессорных вычислительных систем привело к созданию новых принципов описания и управления параллельно протекающими вычислительными процессами с целью их эффективной реализации. Основным понятием, с которым в этом случае приходиться иметь дело пользователю, решившему решать свою сложную прикладную задачу на МВС, является понятие параллельной программы. А сам процесс создания параллельной программы называется параллельным программированием. Предполагая, что параллельная программа это такая программа, которая описывает несколько процессов, работающих совместно над выполнением некоторой задачи, а каждый процесс это последовательность операторов, выполняемых один за другим, тогда суть параллельного программирования сводится к следующему. На основе анализа алгоритма решаемой задачи, необходимо выделить совокупности последовательных операторов, описать их средствами соответствующего языка программирования как параллельные процессы с учетом характера взаимодействия при их совместной реализации на МВС. Само взаимодействие программируется как с применением разделяемых переменных, так и с помощью пересылки сообщений, в зависимости от типа МВС, на которой будет реализовываться процесс решения прикладной задачи. Следует добавить, что при любом виде взаимодействия процессов необходима взаимная синхронизация [9].

Таким образом, создание и реализация параллельной программы требует наличия соответствующих языков параллельного программирования, трансляторов, преобразующих параллельную программу в эффективный для данной МВС код, и операционной системы, управляющей параллельным процессом выполнения задачи.

Существует достаточно большое количество моделей параллельных вычислений, которые используются при создании параллельных программных приложений, но среди них можно выделить пять основных - итеративный параллелизм, рекурсивный параллелизм, «производители и потребители» (конвейеры), «клиенты и серверы» и взаимодействующие равные [9].

Итеративный параллелизм используется, когда в программе есть несколько процессов (часто идентичных), каждый из которых содержит один или несколько циклов. Таким образом, каждый процесс является итеративной программой. Процессы программы работают совместно над решением одной задачи, они взаимодействуют и синхронизируются с помощью разделяемых переменных или передачи сообщений. Итеративный параллелизм часто встречается в научных вычислениях, выполняемых на МВС.

Рекурсивный параллелизм может использоваться, когда в программе есть одна или несколько рекурсивных процедур, и их вызовы независимы,

т.е. каждый из них работает над своей частью общих данных. Рекурсия часто применяется в императивных языках программирования, особенно при реализации алгоритмов типа «разделяй и властвуй» или «перебор с возвращением». Рекурсия является одной из фундаментальных моделей и в символических, логических, функциональных языках программирования. Рекурсивный параллелизм используется для решения таких комбинаторных задач, как сортировка, планирование (задача коммивояжера) и игры.

Производители и потребители - это взаимодействующие процессы. Они часто организуются в конвейер, через который проходит информация. Каждый процесс конвейера является фильтром, который потребляет данные с выхода своего предшественника и производит входные данные для своего последователя. Фильтры встречаются на уровне приложений (оболочки) в операционных системах типа Unix, внутри самих операционных систем, внутри прикладных программ, если один процесс производит выходные данные, которые потребляет (читает) другой процесс.

Клиенты и серверы - это наиболее распространенная модель взаимодействия в распределенных системах, от локальных сетей до World Wide Web. Клиентский процесс запрашивает сервис и ждет ответа. Сервер ожидает от клиентов запросов, а затем действует в соответствии с этими запросами. Сервер может быть реализован как одиночный процесс, который не может обрабатывать одновременно несколько клиентских запросов, или (при необходимости параллельного обслуживания запросов) как многопоточная программа. Клиенты и серверы представляют собой параллельное программное обобщение процедур и их вызовов: сервер выполняет роль процедуры, а клиенты их вызывают. Но если код клиента и код сервера расположены на разных процессорах, обычный вызов процедуры использовать нельзя. Вместо этого необходимо использовать удаленный вызов процедуры или рандеву.

Взаимодействующие равные - эта модель взаимодействия встречается в параллельных программах, в которых несколько процессов для решения задачи выполняют один и тот же код и обмениваются сообщениями. Взаимодействующие равные используются для реализации распределенных параллельных программ, особенно при итеративном параллелизме и децентрализованном принятии решений в распределенных системах.

20.1 Языки параллельного программирования

В настоящее время сформировалось несколько подходов к созданию языков параллельного программирования, среди которых можно выделить следующие. Расширение существующих, широко распространенных языков программирования, таких как Фортран, С. Создание новых, так называемых систем параллельного программирования, ориентированных на определенный класс вычислительных систем, например OpenMP, MPI. И, наконец, третий подход, связанный с созданием новых языков параллельного программирования, не ориентированных на какую либо конкретную архитектуру МВС, но содержащих новые концепции параллельного программирования (ADA, Т-система, НОРМА).

Однако, здесь следует отметить, что многие пользователи понимают, что ни одна система параллельного программирования не гарантирует высокую эффективность вычислительных процессов без скрупулезного изучения их свойств и возможностей. С другой стороны использование традиционных последовательных языков программирования всю дополнительную работу по адаптации программы к архитектуре МВС перекладывает на компилятор, который должен решить задачу автоматического распараллеливания программы [6].

Достоинством расширения существующих языков программирования является их широкое распространение, знакомство с ними пользователей и возможность использования ранее накопленного багажа разработанных программ. Само расширение, чаще всего основано на введении в конструкции языка таких операторов как FORK и JOIN, первый из которых объявляет параллельные процессы, второй инициирует процесс ожидания завершения параллельных процессов. Недостатком такого подхода является необходимость сохранения концепций используемого языка, которые мешают, а в некоторых случаях затрудняют создание языковых средств параллельного программирования.

Достоинство второго похода заключается в возможности создания достаточно эффективных языковых средств параллельного программирования, но ориентированных либо на системы с общей памятью, либо на системы с распределенной памятью. Недостатком таких языков является, следовательно, их специализация и необходимость их детального изучения пользователями.

Достоинство третьего подхода - возможность введения новых концепций, повышающих эффективность языка программирования и надежность написанных на нем программ. Недостатками являются: непроверенность новых концепций, зачастую сложность создания эффективных объектных программ при трансляции с такого языка и, конечно, необходимость изучения пользователем не только синтаксиса языка, но и заложенных в него концепций.

Рассматривая особенности языков параллельного программирования, созданных на основе разных подходов, можно выделить языковые средства

параллельного программирования, типичные для определенного вида структуры МВС. Влияние структуры МВС на языковые средства параллельного программирования показано в таблице . В этой таблице знаком «+» отмечены средства, которые необходимо вводить в языки параллельного программирования, предназначенные для соответствующих ВС, отличительной особенностью которых, в этом случае, является принцип обработки данных - синхронность параллельной обработки. ВС разделяются на два вида: с синхронной параллельной обработкой и с асинхронной параллельной обработкой.

Таблица 20.1 Функции параллельных языков

Тип ВС	**Языковые средства**					
	Синхронная параллельная обработка	**Асинхронная параллельная обработка**				
	Описание массивов и векторов с точки зрения параллельной обработки	**Описание процессов**	**Синхрони зи-рующие примитивы**	**Индексация и завершение паралл-х процессов**	**Синхрониз-я доступа к разделяемым ресурсам**	**Осуществл-е обмена сообщениями**
ВС на основе матричных и векторных процессоров	+	-	-	-	-	-
ВС с ОП	-	+	+	+	+	-
ВС с РП	-	+	-	+	-	+

Синхронная параллельная обработка может выполняться на матричных и векторных процессорах. В таком случае в языки параллельного программирования вводятся операции над массивами как средство укрупнения объектов, над которыми производятся вычисления. При этом, если для векторных процессоров описание массивов не отличается от традиционных, то для матричных процессоров необходимо применять специальные способы размещения массивов, поскольку каждый процессорный элемент системы может работать только с данными, находящимися в его «собственной » памяти. Например, для того чтобы

можно было одновременно выполнять операции над всеми элементами i-й строки и j-го столбца, применяется специальный способ размещения матриц в памяти, получивший название «скошенный».

В некоторых языках вводится специальный набор векторных операций, которые реализуют следующие функции: генерацию вектора целых чисел с шагом 1 по заданным верхней и нижней границам, генерацию вектора заданной длины из одинаковых элементов, замену значений всех компонент вектора на заданное значение, определение длины вектора, выборку компонент вектора по индексу, выборку нескольких компонент по нескольким индексам, циклический сдвиг вектора.

Вводится также набор операций над множествами: определение числа элементов множества, выдача одного из элементов множества, пересечение множеств, объединение множеств, сумма и разность множеств и т.д.

Следует, однако привести пример такого языка как АДА, в котором нет никаких стандартных средств для синхронных параллельных вычислений , но есть возможность создавать типы операций и данных по желанию программиста. Это позволяет создавать языковые конструкции, реализующие параллельные вычисления на конкретных вычислительных системах, хотя при таком подходе, как это отмечалось выше, у программиста могут возникать определенные трудности.

Языковые средства описания асинхронных параллельных процессов, которые происходят в многопроцессорных вычислительных системах, включают в себя описания тел процессов и их взаимодействия. Для реализации асинхронного взаимодействия последовательных процессов необходимо организовать доступ к разделяемым ресурсам, инициацию и завершение процессов. Заметим, что инициация и завершение процессов является одним из видов синхронизации.

Каждый последовательный процесс, который может быть выполнен параллельно с другим процессом, может быть описан как задача, процедура, подпрограмма, функция или блок, которые имеют некоторый специальный признак, указывающий на то, что данная часть программы может выполняться параллельно с некоторыми другими частями этой программы. Таким признаком может быть служебное слово, например, PARBEGIN, какие-либо символы, дополняющие уже используемое служебное слово, дополнительные ключевые параметры, наличие которых в списке параметров показывает, что эта часть программы может выполняться параллельно с некоторыми другими частями этой программы. Такими параметрами могут быть, например, время выполнения процесса и его приоритет.

Каждый процесс, выполняемый на некотором процессоре МВС, может иметь ресурсы, которыми он владеет монопольно, и ресурсы, которые он разделяет с другими процессами, выполняемыми параллельно

на других процессорах этой МВС. На логическом уровне такими разделяемыми ресурсами являются, например, данные. Задача синхронизации заключается в организации доступа параллельных процессов к разделяемым ресурсам таким образом, чтобы они не искажали результаты вычислений.

Простейшими языковыми механизмами синхронизации являются семафоры, введенные Дейкстрой. Семафор - это неотрицательная переменная, на которой определены две операции - P и V (p - первая буква датского слова passeren, означающего «пропустить», а V - первая буква слова vrygeven - «освободить»). Если семафор может принимать только два значения 0 или 1, то он называется двоичным. Операция P на семафоре S выполняется следующим образом. Проверяется значение S. Если S>0, то S:=S-1 и операция P считается завершенной; если S=0, то значение S не изменяется и операция P не завершается до тех пор, пока с помощью операции V значение S не станет больше нуля. Операция V изменяет значение семафора S:=S+1.

Языковые средства инициации процесса, как правило, синтаксически близки к языковым средствам обращения к подпрограммам или процедурам в данном языке, т.е. асинхронный параллельный процесс рассматривается как подпрограмма или функция, но выполняющаяся параллельно с вызывающей программой. Однако несмотря на то, что синтаксически оператор инициации процесса близок к оператору обращения к подпрограмме, семантически (в реализации) они различаются достаточно сильно и время инициации процесса во много раз больше, чем время обращения к подпрограмме. Нормальное завершение всех параллельных процессов, как правило, возникает при выполнении оператора типа STOP. Завершение вызванного процесса и всех порожденных им параллельных процессов происходит при достижении им такого оператора как, например END.

Языковые средства синхронизации доступа к разделяемым ресурсам в многопроцессорных системах с общей памятью основаны на применении механизмов взаимного исключения, аналогичных рассмотренным выше семафорам. Одной из основных языковых конструкций, которая реализует такой механизм является так называемая критическая секция. Критическая секция является последовательностью операторов, имеющих доступ к некоторому разделяемому ресурсу. Ей может быть, например, группа операторов процесса, обрабатывающего массив, который может выполняться параллельно с некоторым другим параллельным процессом, заполняющим этот массив. При работе с критическими секциями устанавливаются следующие правила:

1. Если критическая секция свободна, процесс может войти в неё без каких либо задержек.

2. Когда процесс находится внутри критической секции, другие процессы, которые пытаются войти в ту же критическую секцию, задерживаются перед входом в неё.
3. Когда процесс покидает критическую секцию и есть процессы, ожидающие входа в эту секцию, то одному из них разрешается войти в неё.
4. Дисциплина выбора процессов из очереди, ожидающих входа в критическую секцию определяется характеристиками вычислительной системы и особенностями класса прикладных задач, на которые рассчитана эта МВС.

Сам вход процессов в критическую секцию может быть организован с помощью семафоров.

Характерной особенностью МВС с распределенной памятью является отсутствие единого адресного пространства и для обмена данными между параллельными процессами используется явная передача сообщений через коммуникационную среду. Отдельные процессы описываются с помощью традиционных языков программирования, а для организации их взаимодействия вводятся дополнительные функции. По этой причине практически все системы программирования, основанные на явной передаче сообщений, существуют не в виде новых языков, а в виде интерфейсов и библиотек.

20.2 Особенности трансляторов параллельных систем

Для создания эффективных кодов (для конкретного типа ВС) в трансляторы вводятся специальные функции:

1. *Конвейеризация процесса трансляции.*
   Процесс трансляции, как правило, состоит их 3-х последовательно выполняемых частей: лексический анализ, синтаксический анализ и синтез. Таким образом, эти стадии трансляции можно конвейеризовать, выполняя их на разных процессорах (например, лексический анализ на одном процессоре, синтаксический – на другом и т.д.)
2. *Использование новых алгоритмов трансляции.*
   Можно разбить весь процесс трансляции на отдельные части и выполнять их параллельно на разных процессорах, после чего собрать все результаты обработки в единый программный код.
3. *Анализ семантики программы*
   Разбиение программы на части для параллельного выполнения трансляции на отдельных процессорах осуществляется с анализом связей в программе с целью их минимизации между частями программы при ее разбиении.

Функции трансляции:

I. Функции трансляторов, обеспечивающие автоматическое распараллеливание программ:
   1. Распараллеливание линейных участков;
   2. Преобразование последовательных циклов в векторные операции;
   3. Разбиение программы на синхронные параллельно выполняемые процессы.

II. Функции построения алгоритмов трансляции многопроцессорных ВС:
   1. Конвейеризация процессов трансляции;
   2. Параллельно-последовательный анализ;
   3. Параллельно-последовательная трансляция.

III. Семантические подпрограммы, реализующие языковые конструкции параллельной работы программ:
   1. Создание средств управления асинхронными параллельными процессами;
   2. Создание средств синхронизации доступа к разделяемым ресурсам;
   3. Создание средств обмена сообщениями.

Соответствие функций трансляции структуре ВС приведено в следующей табл.20.2:

Таблица 20.2 Функции трансляторов

Группа ф-й / Тип ВС	I			II			III		
	1	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
ВС на основе матричных и векторных процессоров	-	+	-	-	+	-	-	-	-
ВС с ОП	-	-	+	-	-	-	+	-	+
ВС с РП	-	-	+	+	+	+	+	+	-
ВС с процессорами, содерж. неск. функц-х ИУ	+	-	-	-	-	-	-	-	-

20.3 Операционные системы

Функции ОС:

1. Инициализация и завершение параллельных процессов;
2. Обмен сообщениями между параллельными процессами;
3. Синхронизация параллельных процессов;
4. Распределение ресурсов для самой ОС;
5. Распределение ресурсов между процессами пользователя;
6. Устранение тупиковых ситуаций;

7. Слежение за временем отклика системы

Соответствие функций распределенных ОС структуре ВС приведено в следующей табл.:

Таблица 20.3

Ф-ия ОС / Тип ВС	**1**	**2**	**3**	**4**	**5**	**6**	**7**
ВС с одним центр. ПР и несколькими периф. ПР	-	-	-	+	-	-	+
ВС с несколькими однородными ПР и ОП	+	-	+	+	+	+	-
ВС с несколькими периферий-ными ПР и ОП	+	-	+	+	+	+	+
ВС с РП	+	+	+	+	+	+	+
ВС на основе матричных и векторных процессоров	-	-	-	-	+	-	+

Лекция 19

21. Процессоры высокопроизводительных вычислительных систем

Современные процессоры для высокопроизводительных вычислительных систем (High Performance Computing – HPC) можно разделить на три группы – векторные, которым посвящен отдельный раздел курса лекций, высокопроизводительные универсальные, которые рассматриваются далее, и специализированные, чаще всего применяемые в системах в совокупности с универсальными процессорами. Среди специализированных процессоров следует отметить такие как программируемые графические процессоры (Graphical Processor Unit –GPU), ускорители вычислений с плавающей запятой, программируемые логические интегральные схемы.

На протяжении длительного периода времени прогресс в области создания высокопроизводительных универсальных микропроцессоров отождествлялся со значением тактовой частоты и числе транзисторов на кристалле. Поэтому многопроцессорные высокопроизводительные вычислительные системы различались по мощности процессора, основного её компонента, как мелкозернистые (процессор мог выполнять всего несколько команд) и крупнозернистые; по топологии как RISC (Reduced Instruction Set Computer – компьютер с сокращенным набором команд) и CISC (Complex Instruction Set Computer – компьютер со сложным набором команд) процессоры. Основные отличия между CISC и RISC процессорами кроются в различном количестве и уровнях сложности машинных команд. В классических реализациях архитектуры CISC их много – около или более 100, и они зачастую семантически нагружены, аналогично операторам высокоуровневых языков программирования. RISC команды много проще и их всего порядка 30 и, как следствие, для эффективного выполнения требуют весьма скрупулезной оптимизации первичного программного кода.

Преимущества RISC-процессоров:

– небольшой формат команды;

– команды простые;

– доля устройства управления на кристалле на порядок меньше, чем у CISC-процессоров (команды простые и каждая выполняется за 1 машинный цикл). Поэтому срок проектирования таких процессоров короткий и, следовательно, они дешевые.

RISC системы могут иметь производительность, сравнимую или превосходящую производительность систем с большим набором команд за счет простоты и регулярности структуры команд, позволяющих выполнять комбинацию из нескольких команд быстрее, чем одну эквивалентную этой комбинации сложную команду. Большинство операций в RISC процессорах имеет характер «регистр-регистр», поэтому они имеют большое число регистров общего назначения, а обращения к основной памяти происходит

только для выполнения простых операций загрузки в регистры и занесения в память, тем самым повышается скорость простых операций.

Недостатки RISC-процессоров:

– емкость памяти для хранения программного кода значительно больше;

– большая занятость коммутационной сети, так как вследствие больших объемов программного кода чаще требуется подкачка страниц памяти.

Поэтому при выполнении прикладных программ с большим числом сложных операций, например, с плавающей точкой, эффективность RISC систем становится низкой.

Следует отметить, что грань между архитектурами RISC и CISC становится все менее четкой и в ближайшее время очевидно исчезнет.

Благодаря развитию микропроцессорных технологий сначала практически вся логика и быстрая память компьютера оказалась собранной в одном кристалле, а затем и несколько процессоров оказалось там же. Появились многоядерные процессоры и конструирование высокопроизводительных вычислительных систем превратилось в «игру в кубики». Однако, несмотря на существующий интерес к многоядерным и многопотоковым процессорам, следует обратить внимание на то, что ключевой момент в повышении эффективности применения таких систем заключается не в собственно процессорах, а в необходимых радикальных изменениях в программном обеспечении.

Тем не менее архитектура современного процессора как элемента вычислительной системы обладает следующими особенностями.

1. Большое количество регистров.
2. Масштабируемость структуры до большего числа функциональных узлов.
3. Большая емкость многоуровневой кэш памяти.
4. Явный параллелизм в машинном коде. Поиск зависимостей между командами производит либо аппаратура процессора, либо компилятор, либо совокупность того и другого.
5. Предикация (Predication). Команды из разных ветвей условного ветвления снабжаются предикатными полями (полями условий) и запускаются параллельно. Например, если в исходной программе встречается условное ветвление (по статистике- через каждые 6 команд), то команды из разных ветвей помечаются разными предикатными регистрами (команды для этого имеют предикатные поля), далее они выполняются параллельно, но их результаты не записываются, пока значения предикатных регистров не определены. Когда, наконец, определяется условие ветвления, предикатный регистр, соответствующий «правильной» ветви, устанавливается в 1, а другой в 0. Перед записью результатов процессор будет проверять предикатное поле и записывать результаты только тех команд,

предикатное поле которых содержит предикатный регистр, установленный в 1.

6. Загрузка по предположению (Speculative loading). Данные из медленной основной памяти загружаются заранее. Компилятор перемещает команды загрузки данных из памяти так, чтобы они выполнялись как можно раньше. Следовательно, когда данные из памяти понадобятся какой-либо команде, процессор не будет простаивать. Перемещаемые таким образом команды называются командами загрузки по предположению и помечаются особым образом. А непосредственно перед командой, использующей загружаемые по предположению данные, компилятор вставит команду проверки предположения.

Следует обратить внимание на то, что при создании 2-х, 4-х и более ядерных процессоров, проблема медленной основной памяти всплывает с новой силой. По статистике, если, например, одноядерный процессор 20% своего времени простаивал, ожидая данные из оперативной памяти, то двухядерный будет простаивать 33% времени, а четырехядерный -50%.

22. Предшественники многоядерных процессоров

Как было отмечено выше, можно отождествлять производительность процессора со скоростью выполнения им операций программного кода. В этом случае производительность процессора определяется числом операций, выполненных за единицу времени (секунду). Так как одной из существенных характеристик процессора является его тактовая частота (однако, следует отметить, что для многоядерных процессоров компания Intel отказалась от указания в названии процессора его тактовой частоты), то появилась необходимость связать с ней производительность процессора. Это можно сделать, если рассматривать производительность как произведение количества операций n, выполняемых за один такт процессора (Instruction Per Clock — IPC), на количество тактов m процессора в единицу времени (тактовая частота процессора — F). Таким образом, преобразуя формулу (1) получим:

$$E = n/t = (n/m) \times (m/t) = \text{IPC} \times \text{F}. \qquad (4)$$

Эта формула определяет два подхода к увеличению производительности процессора, первый из которых определяется увеличением тактовой частоты процессора, а второй увеличением числа операций программного кода, выполняемых за один такт процессора. Последнее десятилетие отмечалось использованием комбинации обоих подходов. Однако наращивание тактовой частоты имеет технологические ограничения, которые в последнее время привели к тому, что темпы роста тактовых частот процессоров значительно снизились. При этом рост производительности не прямо пропорционален росту тактовой частоты и ее

дальнейшее увеличение становится нерентабельным. Кроме этого увеличение тактовой частоты приводит к увеличению энергопотребления и, следовательно, к росту выделяемого тепла, с которым современные системы воздушного охлаждения справляются не достаточно эффективно.

Второй подход заключается в увеличении числа функциональных исполнительных устройств (ФУ) внутри самого процессора, что дает возможность одновременно выполнять несколько операций, то есть организовывать параллелизм на уровне операций (Instruction Level Parallelism — ILP). Процессоры, реализующие эту возможность, называются суперскалярными. Практически все современные процессоры это суперскалярные процессоры, ФУ которых представляют собой устройства с конвейерной организацией так, что процесс выполнения любой команды программного кода разбивается на этапы, выполняющиеся на специализированных устройствах. Разработка таких процессоров с несколькими ФУ конвейерного типа, таких как целочисленное арифметико-логическое устройство (АЛУ) (а в ряде случаев их может быть более одного), АЛУ с плавающей точкой и другие потенциально обеспечивают производительность, пропорциональную суммарной производительности используемых ФУ. Однако, простои устройств, связанные с такими факторами как: ожидание завершения обмена с более медленной памятью, выполнение команд условного перехода, зависимости между командами по данным и используемым ресурсам, приводят к тому, что по результатам тестирования таких процессоров, ФУ в лучшем случае загружены производительной работой не более чем на 35%. Поэтому увеличение числа ФУ в процессоре приводит к еще меньшему значению коэффициента их загруженности, несмотря на попытки улучшения архитектурной организации процессора и качества компиляторов. В связи с этим понятен интерес разработчиков процессоров, обративших внимание на создание многоядерных процессоров. Но поскольку каждое ядро процессора это по существу тот же суперскалярный процессор, то необходимо более подробно остановиться на особенностях их построения в хронологической последовательности.

Одним из первых процессоров, который в полной мере обеспечивал параллелизм на уровне операций, являлся процессор многопроцессорного вычислительного комплекса (МВК) «Эльбрус-3», выполненный на множестве интегральных микросхем [9]. В данном процессоре использована архитектура с очень длинным командным словом (Very Long Instruction Word -VLIW) фиксированной длины в 320 бит, обеспечивающая микрораспараллеливание на уровне операций как при скалярных, так и при векторных вычислениях. Каждое командное слово (команда) максимально запускает семь арифметических операций, которые используют в качестве операндов либо результаты ранее выполненных операций, поступающие непосредственно с выходов арифметических устройств, либо числа из быстрой регистровой памяти. В этой же команде предусмотрено:

- управление записью результатов в буфер стека (если эта запись необходима);
- параллельное выполнение условного и безусловного перехода;
- параллельное обращение по восьми каналам в локальную и (или) глобальную оперативную память.

В состав процессора входят девять синхронных исполнительных устройств — пять арифметических (два сумматора, два умножителя, один делитель), два логических, два считывания-записи значений и формирования адресов и два асинхронных исполнительных устройства (планировщик доступа в оперативную память и устройство подготовки и выполнения процедурных переходов). Арифметические устройства выполнены конвейерными так, чтобы обеспечить возможность загрузки каждого устройства каждый такт процессора. Идеология управления каждым тактом реализована только внутри каждой процедуры, т.е. внутри каждой процедуры транслятором составлено полное расписание по тактам работы и переведено в большие команды, составляющие код данной процедуры. Поэтому характер программирования на языках высокого уровня должен быть таким, чтобы процедура была бы независимым модулем, который может независимо транслироваться и независимо вызываться. В этом случае значительная часть работы по оптимизации распределения ресурсов возлагается на оптимизирующий транслятор. Более подробно о МВК «Эльбрус-3» излагается в разделе 24.

22.1 Особенности построения Elbrus 3M

Воплощение архитектуры VLIW в одном кристалле отечественными разработчиками во главе с Б.А.Бабаяном была предпринята при создании сначала микропроцессора «Эльбрус Е2К», а затем его модификации Elbrus 3M [10]. По существу эти процессоры являются в некотором роде прообразами двухядерных процессоров. На рис. 2.1 представлена структурная схема Elbrus 3M. Отличительной особенностью этого процессора является то, что в нем применена архитектура VLIW с командой переменной длины. При этом команда состоит из слогов длиной 32 разряда каждый. Число этих слогов может меняться от 2 до 16, таким образом, длина команды может достигать 512 бит. Это приводит к значительному уменьшению затрат емкости памяти для хранения программного кода.

Функциональные устройства процессора разнесены по двум кластерам (именно это положение дает право считать, что можно было назвать кластер ядром процессора). Здесь кластером называется устройство обработки данных, состоящее из трех одинаковых целочисленных конвейерных АЛУ (правда, один из кластеров имеет также ФУ деления — целочисленного и с плавающей запятой). Кроме этого, имеется 4 канала для данных с плавающей запятой, по 2 на кластер. В каждом канале может выполняться команда типа MADD — «умножить и сложить», что дает темп 8 результатов с плавающей

запятой за такт. Эта команда осуществляется в форме последовательного выполнения умножения и сложения, причем результат первой части команды не помещается в регистр, а подается как вход второй части команды. Такое «зацепление» реализовано еще в супер-ЭВМ Cray-1. Следует также отметить, что набор команд процессора содержит также и четырехадресные команды, например, типа $d = a + b + c$.

В каждом кластере представлены также адресные сумматоры, с помощью которых каждый кластер может одновременно выполнять до 2 операций загрузки регистров или 1 операцию записи в оперативную память. Каждый кластер содержит по 256 регистров, общих для данных целочисленных и с плавающей точкой. Всего в этом процессоре имеется 30 регистровых портов — 20 портов чтения (по 10 на кластер) и 10 портов записи.

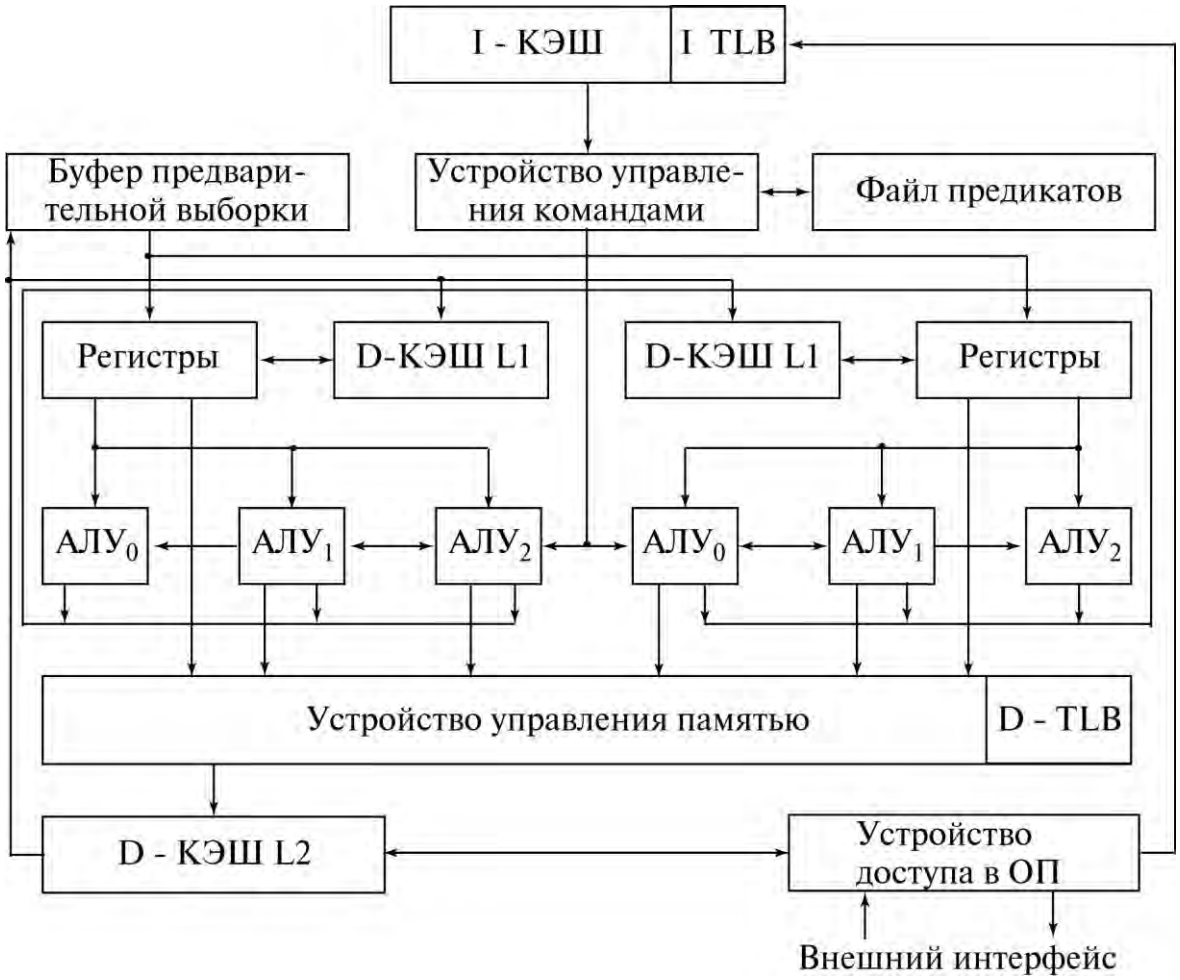


Рис.22.1. Структурная схема Elbrus 3M

В микросхеме процессора предусмотрено размещение двух уровней кэш памяти данных. Кэш данных первого уровня L1 емкостью 8 Кбайт является прямоадресуемым и продублирован в каждом из кластеров и оба кэша имеют по два порта. Данный кэш использует алгоритм сквозной записи данных, что, вероятно, вызвано наличием кэша второго уровня, и является не

блокирующим. Время доступа в кэш первого уровня равно 2 тактам. Как утверждают разработчики, непопадание в кэш данных первого уровня никогда не останавливает работу процессора. Этот кэш имеет буфер адресов отсутствующих строк кэша. Обычно кэш данных второго уровня L2 успевает заполнить кэш первого уровня до того, как этот буфер переполнится. Если же это все-таки произойдет, новые запросы к памяти будут направлены прямо в кэш второго уровня.

Кэш данных второго уровня имеет емкость 256 Кбайт при времени доступа в 8 тактов. Он является двухканальным частично-ассоциативным и имеет 4 банка, то есть обеспечивает 4-кратное расслоение кэш памяти. В отличие от кэша первого уровня, в кэш данных второго уровня применяется алгоритм обратной записи и он является также не блокирующим. При непопадании в кэш второго уровня будет происходить обращение к внешнему интерфейсу оперативной памяти, причем «системная шина» может управлять до 64 запросами для различных строк кэша второго уровня, что является очень высоким показателем.

Как и сама кэш-память, буфер быстрой переадресации (TLB) имеет двухуровневую иерархию: 16 строк полностью ассоциативной памяти плюс 512 строк TLB второго уровня (последний является 4-канальным частично-ассоциативным со временем доступа 2 такта). При этом поддерживается до 4 одновременных преобразований адресов при обращении в кэш второго уровня или к оперативной памяти. При непопадании в TLB первого уровня процессор останавливается на 4 такта. В случае непопадания в TLB второго уровня время ожидания процессора составит от 10 до примерно 200 тактов в зависимости от числа требуемых просмотров в таблице страниц и ситуации с попаданием/непопаданием в кэши второго и третьего уровня (последний является внешним по отношению к кристаллу процессора и подсоединяется либо непосредственно к процессору или через коммутатор, выполняющий функции «системной шины»). Отметим, что при работе с кэшем первого уровня TLB вообще не нужен, так как этот кэш индексируется и тегируется виртуально (в отличие от кэша второго уровня). Следует отметить, что в процессорах типа Эльбрус применена теговая архитектура, при которой каждый элемент данных сопровождается тегом — признаком типа данных, располагаемых в специальных дополнительных разрядах слова данных.

Учитывая большую длину команды, разработчики особое внимание обратили на пропускную способность кэша команд первого уровня (I-кэш), имеющего емкость 64 Кбайт при длине строки 256 байт. Это кэш является 4-канальным частично ассоциативным. Ширина тракта, по которому команды из кэша команд поступают в устройство управления командами, составляет 256 байт. Однако, максимальная скорость заполнения кэша команд составляет 32 байта за такт, что может создать узкое место в процессоре, если коды не будут почти линейными. Буфер TLB для команд имеет емкость 64 строки и является полностью ассоциативным.

Кроме рассмотренных типов кэш-памяти в процессоре есть специализированный кэш-буфер предварительной выборки, который является частью устройства доступа к массивам (этот блок на рисунке не показан) и задействуется только при работе с массивами в циклах. Его емкость составляет всего 4 Кбайт, и он состоит из двух банков с двумя портами в каждом из них. За один такт в буфер можно считать, следовательно, до 4 слов длиной 8 байт. Буфер организован как очередь FIFO и имеет до 64 зон предварительной выборки. Задача этого буфера — обеспечить возможность предварительной выборки данных заранее, до того, как они реально понадобятся, а в данном случае — опережающей выборки в цикле. Такая выборка, происходящая одновременно с выполнением тела цикла, позволяет скрыть задержки при обращении к оперативной памяти.

В архитектуре процессора для того, чтобы по возможности исключить обычные операции перехода, введены 32 одноразрядных регистра-предиката (файл предикатов). Выполняемая команда может сформировать до 7 предикатов: 4 в операции сравнения в АЛУ и еще 3 в операциях логики с предикатами. Предикаты могут использоваться в канале АЛУ или в канале доступа к массивам; для указания на это используются условные слоги в составе длинной команды, содержащие маски предикатов и ФУ. Всего в этих слогах может кодироваться до 6 предикатов, указывающих на то, нужно ли выполнять соответствующие операции из длинной команды. Если условие перехода удается вычислить до выполнения этого перехода, компилятор стремится применять предикатные вычисления, чтобы обойтись без перехода вообще. Если же это не удается, компилятор порождает явные спекулятивные коды. При этом компилятор порождает коды для обеих ветвей программы, возникающих при условном переходе, и, пользуясь большим числом ФУ и регистров, заставляет процессоры выполнять обе ветви программы (спекулятивные вычисления). При возникновении ситуации исключения результат снабжается тегом недействительного значения.

В заключении следует отметить, что процессоры Эльбрус должны обладать сегментно-страничной организацией и поддержкой мультипрограммирования в стиле x86 и в сочетании с разработанными средствами двоичной компиляции и специальными аппаратными средствами ее поддержки способны выполнять x86 коды.

22.2 Технология Hyper Threading

Рассмотрим следующий вариант организации процессора, являющегося предшественником многоядерных процессоров. К таким процессорам относится Intel Pentium 4 с тактовой частотой 3,06 ГГц и поддержкой технологии Hyper-Threading (HT) [11]. Фактически эта технология, основанная на включении нескольких ФУ в процессоре и, следовательно, позволяющая организовать параллелизм на уровне операций ILP, дает возможность построить два логических процессора в одном физическом.

Процессор Pentium 4 имеет в общей сложности семь ФУ, два из которых могут работать с удвоенной скоростью - две операции за такт. Каждый логический процессор имеет свои регистры (включая свой собственный счетчик команд) и контроллер прерываний (Architecture State — AS). В современных приложениях, как правило, может выполняться не одна, а несколько задач или несколько потоков (threads) одной задачи, называемых также нитями. В этом случае два параллельно выполняемых потока работают со своими собственными независимыми регистрами и прерываниями, но при этом используют одни и те же ресурсы процессора. После активации каждый из логических процессоров может самостоятельно и независимо от другого выполнять свой поток, обрабатывать прерывания или блокироваться. Использование двух логических процессоров позволяет обеспечить параллелизм на уровне потоков, реализованный в современных операционных системах и высокоэффективных приложениях.

Покажем преимущества технологии HT по сравнению с технологией вычислений в процессоре без HT. Рассмотрим некоторый условный процессор, в котором имеется три ФУ: устройство для работы с целыми числами (арифметико-логическое устройство — ALU), устройство для работы с числами с плавающей точкой (FPU) и устройство для записи и чтения данных из памяти (Store/Load — S/L). Допустим, каждая операция осуществляется за один такт работы процессора. Предположим, что выполняется программа, состоящая из трех операций: первые две — арифметические действия с целыми числами, а последняя — сохранение результата. В этом случае для последовательного варианта выполнения программы, она реализуется за три такта процессора (рис. 2.2 *а*). В первом такте задействуется устройство ALU процессора, во втором такте также оно, а в третьем устройство записи и чтения данных из памяти S/L.

Как было отмечено выше, для реализации параллелизма на уровне операций необходимо в процессор вводить дополнительные ФУ. Пусть в нашем условном процессоре используется не одно, а два устройства ALU. Тогда две первые операции с целыми числами можно было бы выполнить одновременно, то есть за один такт. Для этого необходимо, чтобы сами операции были независимыми, то есть результат одной из них не зависел от результата другой. С учетом сделанных предположений этот процессор выполнит указанную программу за два такта (рис. 2.2 *б*).

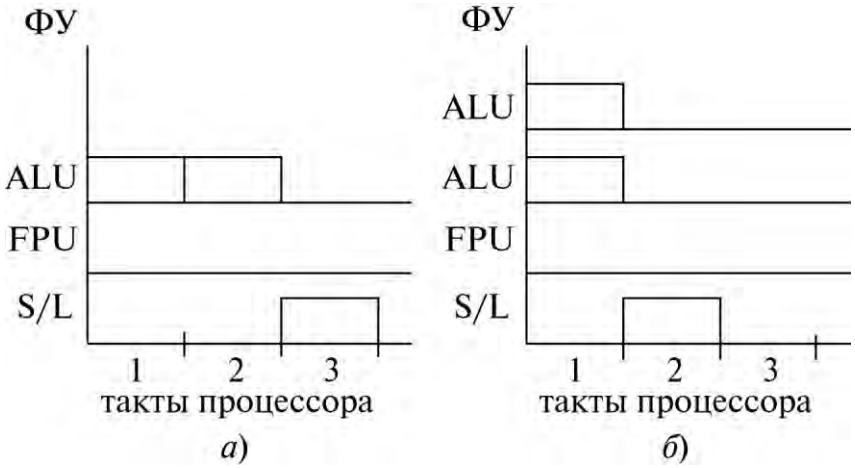


Рис. 22.2. Последовательное (*а*) и параллельное (*б*) выполнение операций с двумя ALU

Рассмотрим теперь случай, когда наш условный процессор выполняет два разных потока A и B. Поток A как и в предыдущем случае включает две независимые операции с использованием устройства ALU и одну операцию по сохранению результата. Поток B выполняет считывание данных из памяти (устройство S/L), операцию с числами с плавающей точкой при помощи устройства FPU и сохранение результата (S/L). Если бы оба потока исполнялись изолированно, то для выполнения первого потока потребовалось бы два такта процессора, а для второго — три (рис. 2.3). При одновременном исполнении двух потоков процессор будет постоянно переключаться между обоими потоками так, что за один такт процессора выполняются только операции какого либо одного из потоков (рис. 2.4).

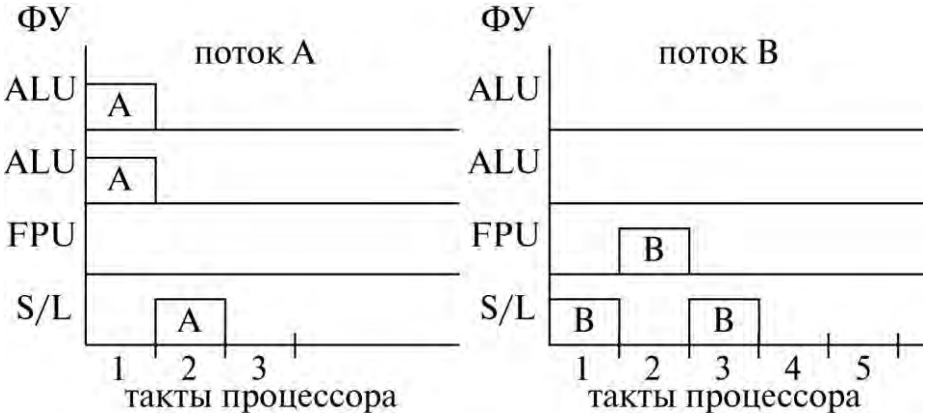


Рис. 22.3. Независимое выполнение потоков A и B

Как видно из рисунков, на каждом такте процессора используются не все функциональные устройства процессора. Для этого нужно по возможности не только выполнять параллельно независимые операции одного потока, но и совместить выполнение операций различных потоков. В нашем примере выполнение двух арифметических операций с целыми числами первого потока можно совместить с загрузкой данных из памяти

второго потока и выполнить все три операции за один такт процессора. Аналогично на втором такте процессора можно совместить операцию сохранения результатов первого потока с операцией с плавающей точкой второго потока (рис. 2.5). Собственно, в таком параллельном выполнении двух потоков и заключается основная идея технологии Hyper-Threading.

Рассмотренная ситуация является достаточно идеализированной и на практике, в зависимости от характеристик конкретной программы, выигрыш от использования технологии HT может быть не столь существенным. Так возможность одновременного выполнения на одном такте процессора операций от разных потоков ограничивается тем, что эти операции могут задействовать одни и те же функциональные устройства процессора.

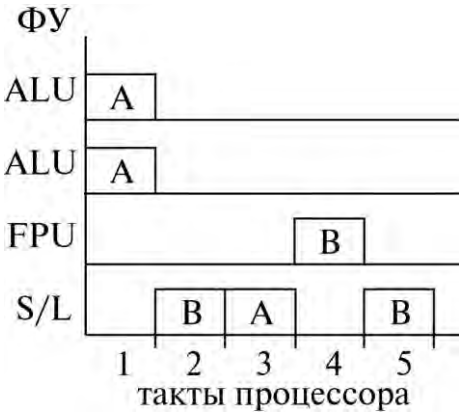


Рис. 22.4. Одновременное выполнение потоков A и B

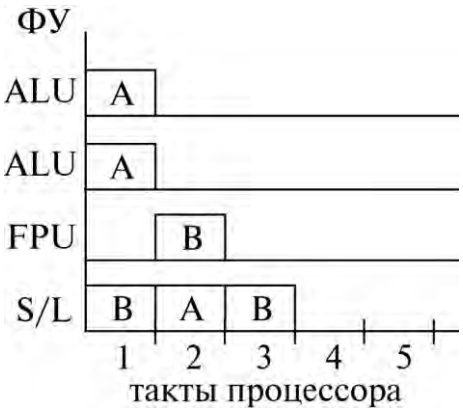


Рис. 22.5. Выполнение потоков A и B с применением технологии HT

Оценим эффективность технологии HT на другом примере потоков операций, выполняемых нашим условным процессором. Пусть имеется два потока операций, каждый из которых по отдельности выполняется за пять тактов процессора (рис. 2.6). При последовательном выполнении потоков (без применения технологии HT) потребовалось бы десять тактов работы процессора. Теперь определим, что произойдет при использовании технологии HT (рис. 2.7). На первом такте каждый из потоков задействует

различные устройства процессора, поэтому выполнение операций легко совместить. То же самое относится и ко второму такту, а вот на третьем такте операции обоих потоков пытаются задействовать одно и тот же ФУ процессора, а именно устройство S/L. В результате возникает конфликтная ситуация, и один из потоков должен ждать освобождения требуемого ресурса процессора. То же самое происходит и на пятом такте. В итоге оба потока выполняются не за пять тактов, а за шесть.

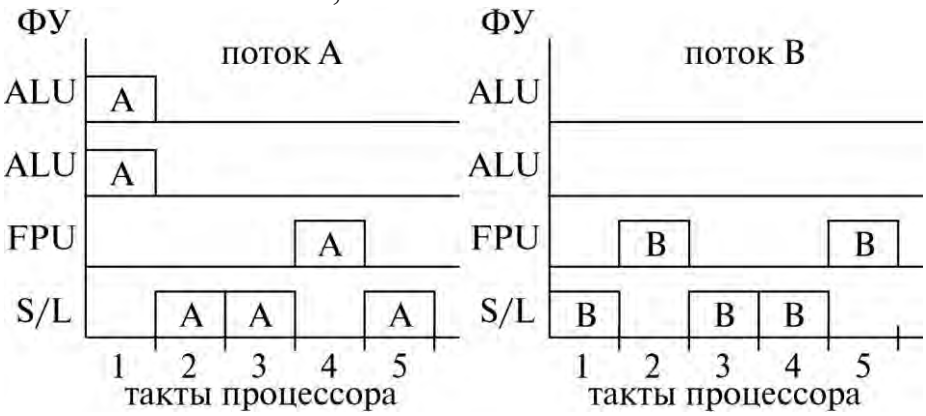


Рис. 22.6. Пример двух потоков операций

Этот простой пример показывает, в каких случаях от технологии HT можно ожидать максимального выигрыша:

- оба потока должны использовать разные устройства процессора;
- отдельные потоки должны быть оптимизированы под конкретный тип процессора (компилятор должен создать такие последовательности команд, для которых число конфликтов из-за использования общих ресурсов было бы минимальным);
- выполняемые потоки должны принадлежать различным приложениям, так как в этом случае велика вероятность того, что будут использоваться различные ресурсы процессора.

Рис.22.7. Возникновение конфликтных ситуаций при применении технологии HT

Приведем еще один пример, который показывает, что не всегда технология HT приводит к увеличению производительности процессора. Рассмотрим наиболее распространенный случай, когда выполняются два одинаковых потока. Как видно из рис. 2.8, в этом случае становится просто невозможным параллельное выполнение операции от разных потоков за один такт процессора, несмотря на применение технологии HT.

Рис. 22.8. Пример, когда применение технологии HT не дает выигрыша в производительности

Избежать недостатков процессоров с технологией Hyper-Threading в большей степени удается, если изолировать в пределах одного процессора выполнение различных потоков операций. Для этого и создаются многоядерные процессоры. Тогда в идеальном варианте каждый поток операций использует отведенное ему ядро процессора, что позволяет избежать конфликтных ситуаций и увеличить производительность процессора за счет параллельного выполнения потоков операций (при этом сохраняя параллелизм ILP внутри ядра процессора). На рис. 2.9 *а—д* представлены временные диаграммы выполнения примера потоков программ для всех рассмотренных случаев их реализации, в том числе и на двухядерном процессоре.

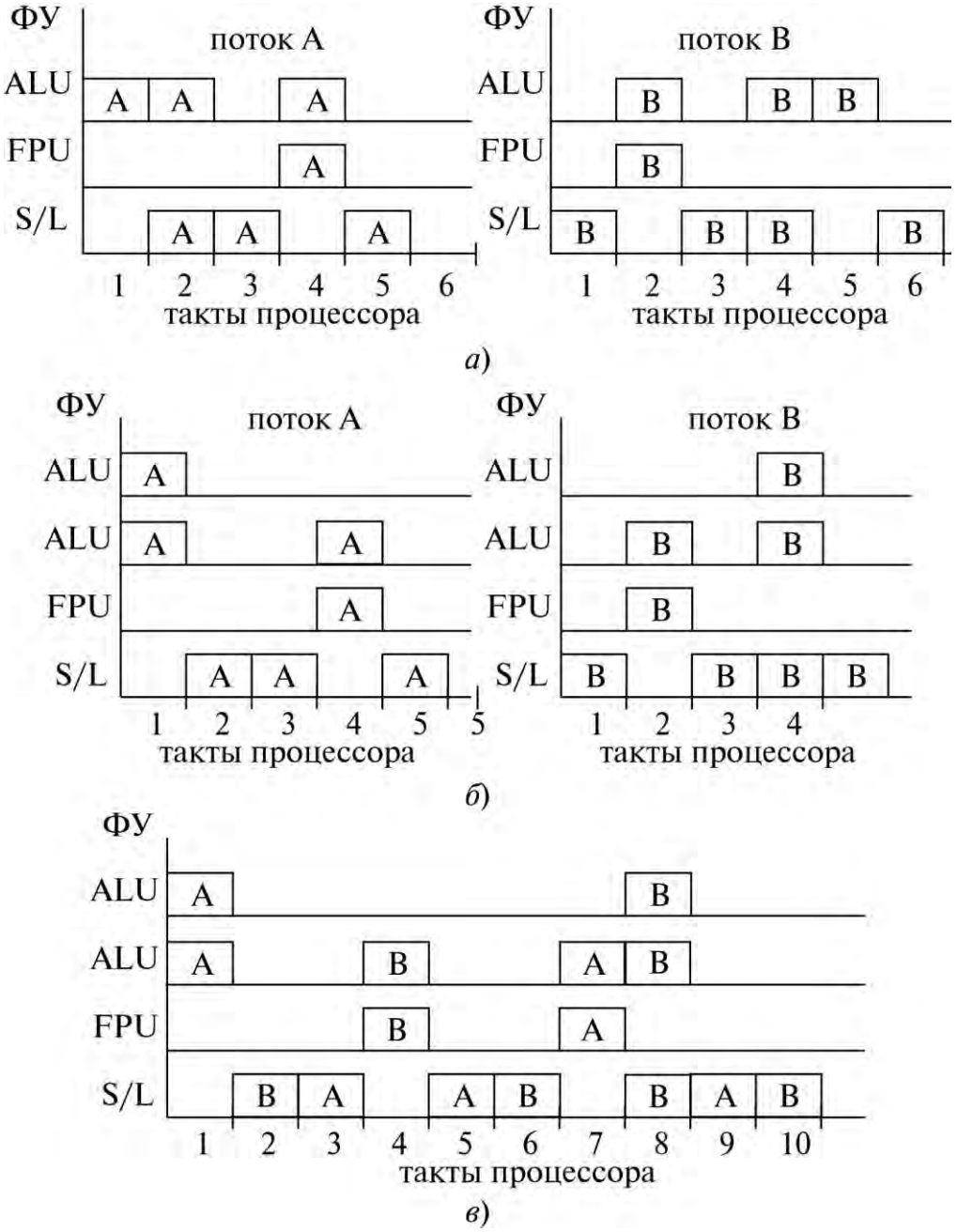


Рис. 22.9. Независимое выполнение двух потоков на процессоре с одним АЛУ (*а*), независимое выполнение двух потоков на процессоре с двумя АЛУ (*б*), одновременное выполнение двух потоков на процессоре без HT (*в*)

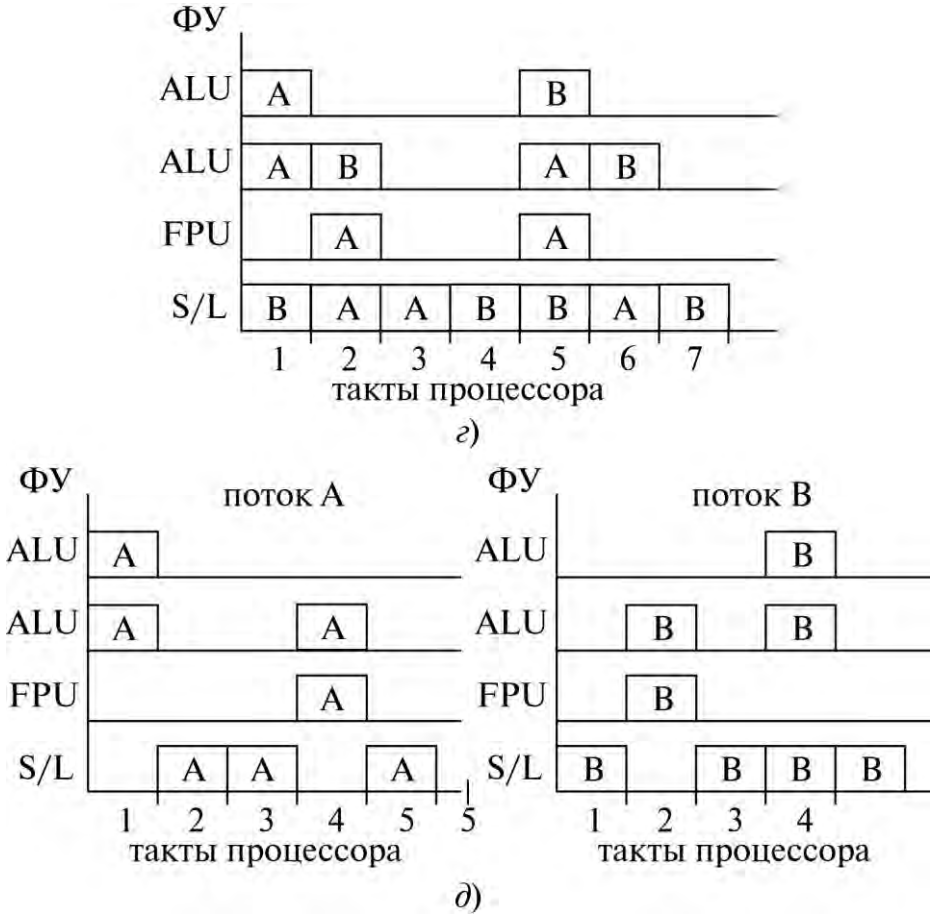


Рис.22.9. Окончание. Выполнение двух потоков на процессоре с HT (*г*), выполнение двух потоков на двухядерном процессоре (*д*)

22.3 Оценка эффективности двухядерного процессора.

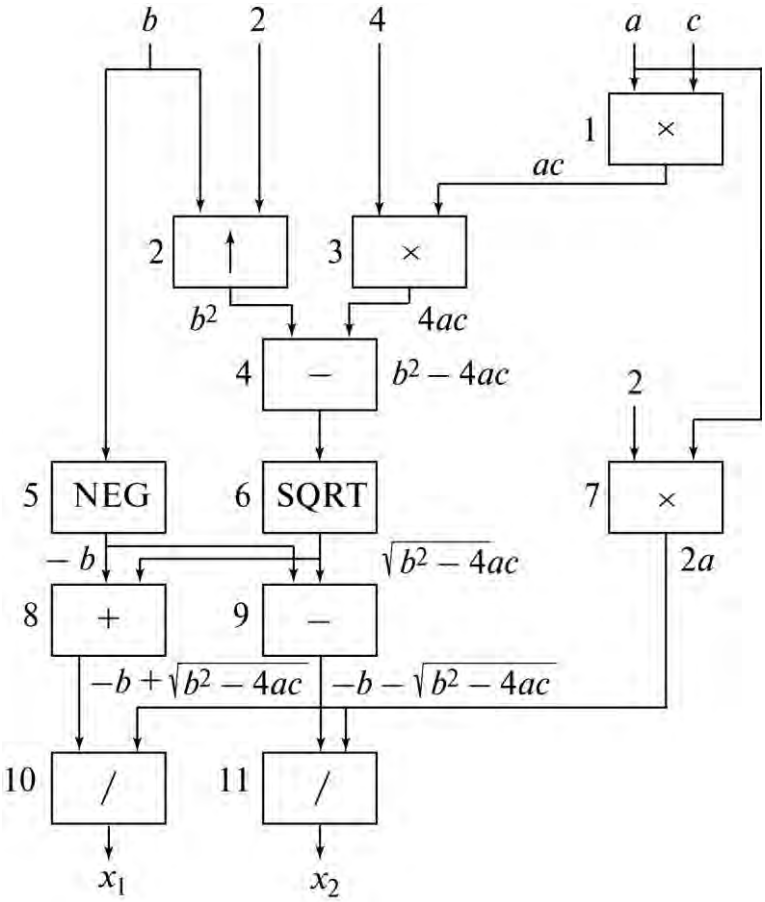
Следует обратить внимание на то, что двухядерные процессоры не всегда в два раза производительнее одноядерных. Причина заключается в том, что для реализации параллельного выполнения двух потоков необходимо, чтобы эти потоки были полностью независимы друг от друга и чтобы операционная система и само приложение поддерживали на программном уровне возможность распараллеливания задач, а компилятор «видел» параллелизм на уровне команд и нитей.

Рассмотрим пример вычислительного процесса — нахождение корней уравнения второго порядка; проходящего в двухядерном процессоре. Как известно, эти корни x_1 и x_2 вычисляются следующим образом:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}. \tag{5}$$

Сетевое представление показано на рис. 2.10. Различные операции представлены прямоугольниками, пронумерованными слева от них. Допустим вычисления x_1 и x_2 происходят в процессоре с одним ALU и все

операции выполняются за один такт, а все исходные данные располагаются в регистрах процессора. В этом случае, компилятор преобразует описание вычислений, представленного на языке высокого уровня, в последовательность команд, например, так как пронумерованы соответствующие операции — 1, 2, …, 11. В этом случае вычисления завершатся за 11 тактов. Если вычисления происходят в двухядерном процессоре при прежних остальных допущениях, то компилятор должен сформировать две последовательности команд (потоков). Например, поток 1 — 1, 3, 4, 6, 9, 11, поток 2 — 2, 5, 7, 8, 10. При этом необходимо учесть задачу синхронизации выполнения потоков, так как каждый из них использует промежуточные результаты друг друга. Так поток 1 может выполнять операции 4, 9, 11 только после получения результатов соответственно операций 2, 5, 7, второго потока, а второй поток операцию 8 может начать только после получения результата операции 6 первого потока. Таким образом, если оба потока выполняются на разных ядрах процессора, и временем передачи промежуточных результатов можно пренебречь (можно считать, что обмен данными между ядрами осуществляется параллельно (совмещается) с обработкой данных), то общее время получения результатов будет равно времени выполнения самого длинного потока. В нашем примере оно будет равным 6 тактам. Тогда коэффициент ускорения выполнения вычислений на двухядерном процессоре будет равен 11/6.



×	×	∑	√	∑	/
1	3	4	6	9	11
2	5	7		8	10
×	*–b*	×		∑	/

Рис.22.10. Сетевое представление процесса вычислений

Лекция 20

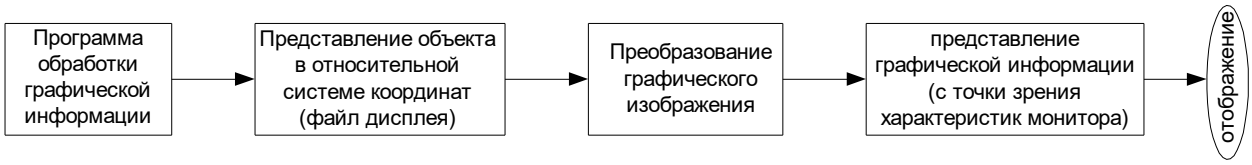
23. Основы обработки графической информации

В последнее время большое внимание уделяется проблеме обработки графической информации во всех её аспектах, в том числе созданию специализированных графических процессоров. Рассмотрим более подробно особенности обработки графической информации, которые определяют свойства создаваемых GPU. Объектом такой обработки является изображение. Изображения обычно задаются цифровой функцией f(i,j) в виде, например, двумерного массива. Элементы цифрового изображения f(i,j) в i-ом столбце и j-й строке (называемые элементами изображения или пикселями) могут быть заданы как бинарные черно-белые (две градации), многоградационные (например, 256 градаций) или в виде многоградационного вектора (например, с 256 градациями по каждой из составляющих – красной, зеленой и синей). В соответствии с этими представлениями изображения называется просто бинарным, полутоновым и многоспектральным. Величина цифрового изображения задается размерностью массива из m столбцов и n строк. Эти характеристики определяют формат данных, с которыми должен работать GPU.

Задачей обработки графической информации является синтез и отображение изображения (например, чертежей), заключающееся в том, что в процессе проектирования какого либо объекта, например механической конструкции, производится отображение на экране монитора под разными углами зрения уже спроектированных участков, и на основе их представления и оценки, выполняются дальнейшие этапы проектирования. Таким образом, функции обработки графической информации должны быть следующими.

1. Оперирование произвольными двумерными и трехмерными изображениями
2. Осуществление операций масштабирования (увеличение и уменьшение), параллельного переноса, поворота и др., формирование проекции (перспективного изображения) для получения эффекта стереоскопичности в случае трехмерных объектов, исключение фрагментов объектов, скрытых за другими изображениями.
3. Представление ощущения цвета и материала на чертеже, создание тонов различной интенсивности.
4. Раздельная реализация функций 2 и 3 по отдельным объектам изображения.
5. Выбор по желанию пользователя произвольного фрагмента из набора объектов и представление его в желаемом месте на экране дисплея, как это делается при работе с большими чертежами.
6. Инвариантность программы обработки графических изображений по отношению к типам мониторов.

Для осуществления указанных функций, при обработки графических изображений, применяется следующая последовательность этапов.

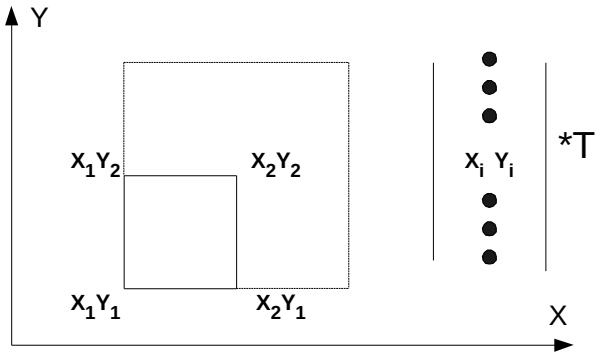


Исходя из функций первых двух блоков (программа обработки графической информации и представление объекта в относительной системе координат) требуется большая ёмкость кэш-памяти, чтобы в ней можно было хранить либо всё изображение целиком, либо его детали.

Функции блока преобразования графической информации:

1. Оперирование производится двумерными и трёхмерными изображениями. Основное требование – это работа с 8, 16 и 32-х битными операндами (это связано с представлением цвета).
2. Выполнение операций масштабирования, поворота, проекции, преобразование координат.
3. Создание представления цвета и материала на чертеже, создание тонов различной интенсивности.

Пример масштабирования:



Для масштабирования используют матрицу масштабирования T.

$T = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$ тогда над каждой точкой примитива выполняется следующая операция:

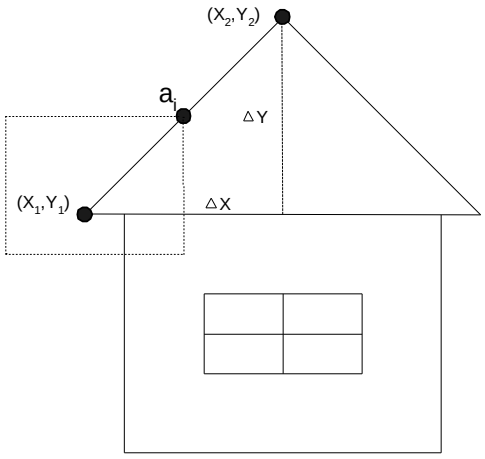
X_{new}= a*x +b*y

Y_{new}= c*x +d*y

Для двумерного случая b=0 и c=0.

Таким образом, базовыми операциями графического процессора должны быть – умножение и сложение. Если учесть, что операция умножения координат многих точек изображения на матрицу T многократно повторяется, то такую обработку можно свести к векторной.

Пример выделения окна



Проблема выделения окна из общего изображения. $(X_1+\Delta X/2;Y_1+\Delta Y/2)$ – таким способом, последовательного выполнения деления на 2, суммирования и сравнения с границей Выделенного окна, ищутся координаты точки a_i.

Получается, что базовой операцией для выделения является деление на 2 или константу и сложение. Чтобы ускорить эту операцию надо хранить константы в специальном регистре.

Пример удаления невидимых линий и невидимых плоскостей

При обработке графических трехмерных изображений важную роль играет функция удаления невидимых линий и невидимых плоскостей для придания представляемому объекту естественного вида. Для этих целей может быть эффективно использован так называемый алгоритм Z – буферизации. Алгоритм заключается в том, что в специальной памяти, называемой Z-буфером, запоминаются все данные о глубине отдельных элементов изображения, представленных на экране монитора (значения по Z координате). Затем производится сравнение значений вновь представленного фрагмента изображения и значений, записанных в Z-буфере для соответствующих точек. Если значения координат оказываются больше чем значения Z, то элемент изображения не виден, а если меньше, то элемент изображения представляется как видимый. В ходе сравнения эти значения координат замещают прежние значения Z. Данный алгоритм требует от процессора большой емкости быстрой памяти и выполнения операции сравнения в векторном режиме.

23.1 Процессор i860. Структурная схема. Регистры

Процессор i860 фирмы Intel впервые был представлен в 1989 г. и затем нашел широкое распространение как компонент многопроцессорных вычислительных систем. Это был первый 64-разрядный процессор, состоящий из 1 млн. транзисторов на одном кристалле. Разработчики заявили его как RISC процессор, хотя он состоял из 3-х основных устройств – центрального процессора, представляющего именно RISC процессор, процессора с плавающей точкой и трехмерного графического процессора. Поэтому авторы называли его супер-ЭВМ Cray-1 в одном кристалле. По существу этот процессор стал прообразом современных суперскалярных и графических процессоров. Благодаря наличию 3-х устройств в i860 и

возможностью их параллельной работы и конвейеризации, одновременно может выполняться три типа действий: целочисленные арифметические операции, сложение и умножение с плавающей точкой. Процессор имеет 64 линии данных, позволяющих осуществлять одновременную передачу 8 байт, 32 –х разрядную адресную шину, что дает возможность адресовать 4 Гбайта памяти. Память имеет страничную организацию, причем размер страницы – 4 Кбайта.

Доступ к данным осуществляется посредством двух встроенных блоков кэш-памяти: один емкостью 4 Кбайта предназначен для команд, а другой емкостью 8 Кбайт для данных. Блок управления кэш-памятью управляет выполнением трансляции адресов с целью обеспечения последующего доступа к данным и командам. Механизм доступа имеет конвейерную организацию, которая позволяет инициировать новый цикл, пока текущий находится еще в двух тактах от своего завершения. Эффективно этот механизм работает с кэш-памятью, однако, если данные находятся в основной памяти, то процесс значительно замедляется.

Структура процессора содержит две основные шины. 64-разрядная шина команд (на рис. 23.1 она разделена на две) позволяет передавать 32-разрядные команды, считанные из кэш-памяти, в каждом такте как в центральный процессор так и в процессор с плавающей точкой. 128-разрядная внутренняя шина данных показана как две 64-разрядные шины.

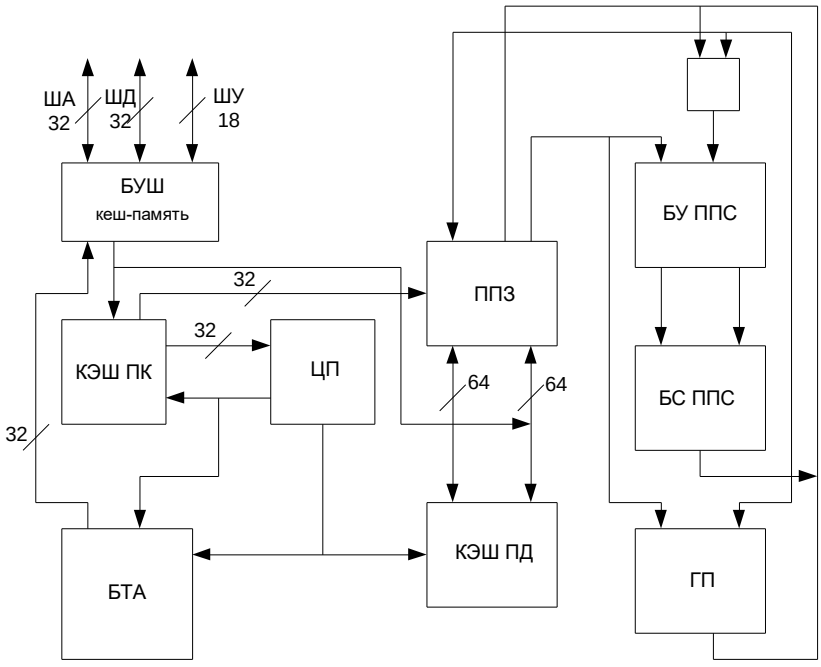


Рис. 23.1 Структурная схема процессора i860

ША – адресная шина 32 разряда

ШД – шина данных 64 разряда
ШУ – шина управления 18 разрядов
БУШ – блок управления шиной и кэш-памятью
БТА – блок трансляции адреса
КЭШ ПК – кэш памяти команд (4кб)
ЦП – центральный процессор. Содержит скалярный RISC процессор. (всего 42 команды)
ППЗ – процессор с плавающей точкой.
БУ ППЗ – блок умножения процессора с плавающей точкой.
БС ППЗ – блок суммирования процессора с плавающей точкой.
ГП – графический процессор.
КЭШ ПД – кэш памяти данных (8кб)

Блоки векторных операций имитируют Cray и поэтому секционированы и реализуют конвейер операций (в векторном режиме) или без него в скалярном режиме.
ГП работает с 8,16,32,64,128 битными единицами информации.
Считается, что это единственный процессор, у которого внутренняя шина данных 128 бит.
Блоки умножения и сложения могут «зацепляться» и это реализовано аппаратно.

В i860 функции выборки и декодирования всего набора команд, независимо от того, к какому типу они относятся, выполняет центральный процессор. Кроме того, он осуществляет операции, относящиеся к следующим категориям:

1. Загрузка и запоминание в целочисленных регистрах операндов, выбранных из памяти;
2. Пересылка операндов из целочисленных регистров в регистры с плавающей точкой;
3. Арифметические операции над 8,16,32 разрядными целыми числам (64-разрядную арифметику поддерживает графический процессор);
4. Сдвиговые и логические операции;
5. Управление пересылками, куда относятся непосредственные и косвенные переходы и команды вызова;
6. Функции управления системой, включающие загрузку регистров управления, работу с шиной и кэш-памятью.

На долю ППЗ остается только произвести необходимые действия над теми операндами, которые для него готовит ЦП. В скалярном режиме (без конвейеризации) на выполнение каждой команды плавающей арифметики уходит от 3-х до 4-х циклов, в случае же конвейера ППЗ выдает результаты по окончании каждого цикла, в крайнем случае через два. Поэтому ППЗ имеет два набора команд.

Скалярные команды:

1. Операции умножения, получения обратной величины, извлечение квадратного корня (все это делает устройство умножения);
2. Операции сложения, вычитания сравнения, преобразование в целое число и округления до целого (их выполняет суммирующее устройство);
3. Операции пересылки в целочисленные регистры.

К конвейерным командам относятся – умножение, сложение, вычитание, преобразование в целое число и округление до целого.

Графический процессор – это специальные аппаратные средства трехмерной графики, которые обеспечивают выполнение таких функций как, например Z-буферизация. В Z-буферах хранится информация о третьей координате каждой точки изображения, благодаря чему имеется возможность высокоскоростного построения трехмерных изображений. ГП фактически использует выходные данные ППЗ. Он оперирует сразу с 64-битными операндами графических данных, не зависимо от того, что сами элементы изображения не бывают длиннее 4-х байтов. Для них определены следующие стандартные форматы:

- в 8-разрядном формате, помимо значений некоторых атрибутов, устанавливаемых в старших битах, несколько младших разрядов предусмотрены для задания уровня интенсивности элемента изображения;
- 16-разрядный формат предусматривает по 6 битов для информации об интенсивности первых двухосновных цветов, как известно, красного и зеленого, а 4 разряда отводятся для третьего – синего. Это объясняется тем, что человеческий глаз менее чувствителен именно к оттенкам синего цвета.
- 32 разрядный формат отводит по три 8-битных поля для каждого из основных цветов, оставляя еще 8 разрядов для передачи некоторых необходимых атрибутов.

Для достижения большего быстродействия и организации многопроцессорного режима предусмотрено использование внешней кэш-памяти второго уровня, которая получила название «когерентной кэш-памяти».

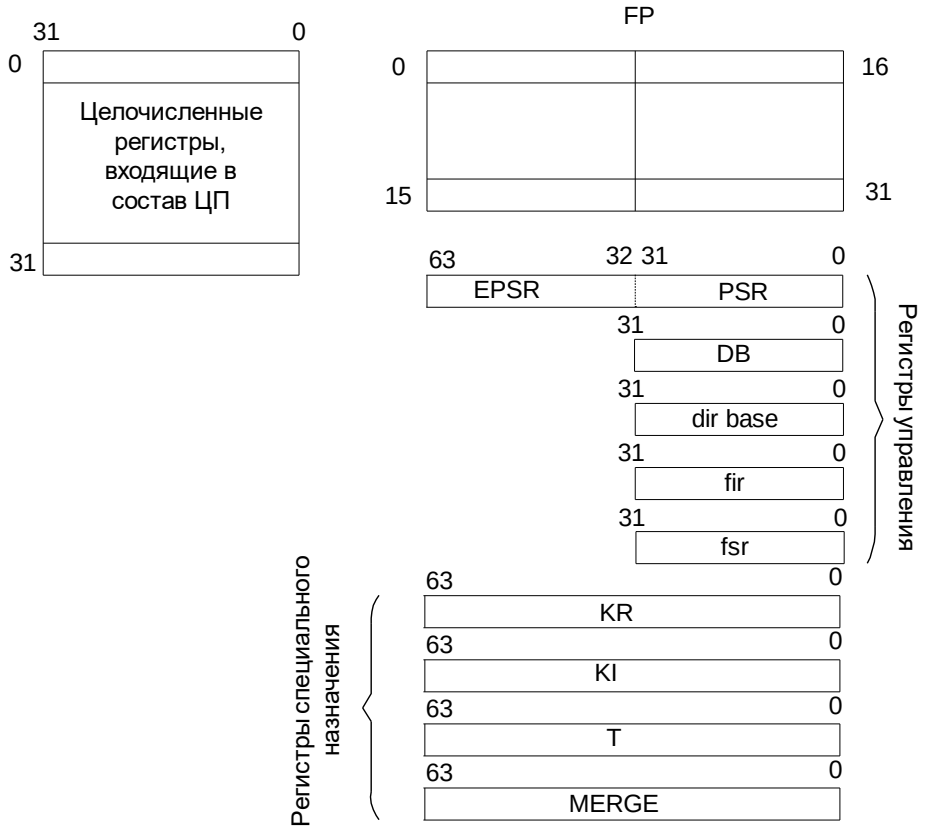
Данные, принятые по внутренней шине могут быть загружены в два набора внутренних регистров (рис. 23.2), которые в терминологии Intel называются «файлами». Файл целочисленных регистров является составной частью центрального RISC процессора и состоит из 32-х 32-разрядных регистров.

В верхнем целочисленном регистре хранится константа «0», которая чаще всего используется при тестировании.

FP – float point регистры (поддерживают ППЗ) могут работать как 32 регистра по 32 разряда, либо как 16 регистра по 64 разряда, или как 8 регистра по 128 разряда.

PSR/EPSR – регистр состояния процессора / расширенный регистр состояния процессора.

DB – регистр прерываний.
dir base – регистр базы каталога (каталог → страница→ база).
fir – регистр ошибочной команды.
fsr – регистр состояния процессора с плавающей точкой.
KR, KI – регистры содержат константы, которые используются в качестве входных данных для устройства умножения, в режиме выполнения сдвоенных инструкций
Регистр T – служит для хранения результата, получаемого с выхода устройства умножения для последующего его использования в качестве входного данного для устройства суммирования (например, для зацепления).
Регистр MERGE – обеспечивает поддержку Z- буферизации (буферизация невидимых точек по оси Z), а также для хранения результатов команд умножения/ сложения и логических операций над элементами изображений.



Регистры (программная модель)

Рис. 23.2 Регистры i860

Лекция 21

24. Вычислительная система «ЭЛЬБРУС»

Исторически «Эльбрусы» авторами определялись как многопроцессорные вычислительные комплексы, однако с точки зрения нашей классификации они относятся к классу МКМД систем с общей памятью.

Основные принципы, положенные в основу проектирования ВС «ЭЛЬБРУС» [9]:

1. Удобство программирования.
2. Эффективная работа аппаратных средств.

Удобство программирования заключается в следующем.

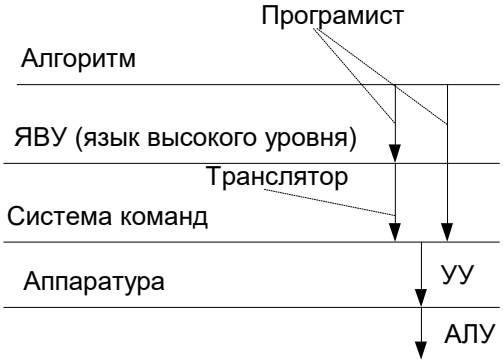
В Эльбрусах используются высокоуровневые языки программирования динамического типа (типы данных могут изменяться в процессе работы) для удобства программирования. Поддержка таких языков программирования может быть аппаратной или программной. При программной поддержке требуется примерно втрое большая ёмкость памяти и вдвое большее быстродействие. Поэтому во всех Эльбрусах **аппаратная** поддержка типов, т.е. система становится теговой (64 разряда данных + 6 разрядов тег). Все семафоры, адреса, данные, данные различных форматов, имена процедур метятся тэгами.

Код операции (например, сложение или умножение) одинаковый для всех типов и длин операндов, а уже в соответствующем модуле (сумматоре или умножителе соответственно) необходимая операция выбирается по тегу (целочисленное сложении, сложение с плавающей запятой т.д.). Это позволяет сократить длину кода операции.

Во всех Эльбрусах **НЕТ** ассемблера. Все программы пишутся на языке высокого уровня (динамического типа) т.к. он поддерживается аппаратно и эффективность такого кода не уступает ассемблеру.

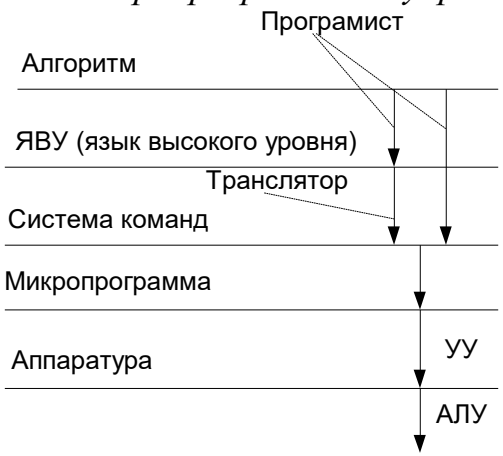
В результате использования теговой архитектуры устраняется семантический барьер. Ниже приводятся схемы информационных моделей доступа программиста к инструментальным средствам программирования для систем обработки данных с жестким управлением, микропрограммным управлением и с теговой архитектурой.

Классическая система (с жёстким управлением).

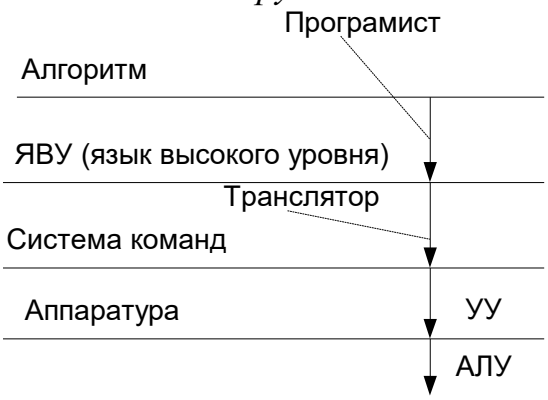


В классической системе программист имеет доступ к ЯВУ и системе команд.

Система с микропрограммным управлением.



Эльбрус-2.



В Эльбрусах - 2,3 программист не имеет доступа к системе команд. Считается, что в Эльбрусе-3 транслятор формирует «микрокоманду». На самом деле – это очень длинное командное слово.

Эффективная работа аппаратных средств заключается в следующем.

Как известно, эффективность работы процессора в основном определяется пропускной способностью иерархически организованной памяти. Введение многоуровневой кэш-памяти, кроме положительных моментов имеет

и отрицательные. Например, если программа работает с большим массивом, который полностью не помещается в кэш-память (а это наиболее типичная ситуация при решении сложных прикладных задач), то происходит следующее. Обычно работа с массивом заключается в последовательном обращении к его элементам с шагом 1 или некоторой константой. В этом случае кэш-память бесполезна, т.к. любой считанный элемент массива либо никогда не затребуется повторно, либо затребуется после обращения ко всем остальным элементам массива, когда он этими обращениями будет вытеснен из кэш-памяти. Более того, т.к. обмен между уровнями памяти осуществляется блоками, может произойти вытеснение и отдельных локальных данных процедуры, в которой размещен цикл. В результате такая работа с массивами как векторные вычисления реализуется не эффективно. В соответствии с этим в «Эльбрус-2» и с учетом языковой семантики проводимых вычислений сверхоперативная память (в современном определении кэш-память) разделяется на 4 функционально специализированные части:

- буфер команд – 256 слов
- буфер стеков операндов – 256 слов
- буфер массивов – 256 слов
- ассоциативное ЗУ (АЗУ) глобалов – 1024 слова

Буфер команд содержит команды исполняемой программы. Код программы постоянная информация, не изменяемая в процессе счета, поэтому команды при использовании не отсылаются обратно в оперативную память.

Буфер стека содержит локальные данные вызванных процедур, информация из него откачивается в оперативную память (в продолжение стека в памяти) в соответствии со стековой дисциплиной, которой подчинены области данных вызываемых процедур в языках высокого уровня.

Буфер массивов предназначен для хранения элементов массива, обрабатываемых в цикле.

В АЗУ глобалов размещается информация, не попадающая по своему характеру в остальные буферные памяти. Это преимущественно глобальные данные (глобальные не только с точки зрения процессоров, но и с точки зрения процедур).

Благодаря всему этому доля времени простоя процессора сведена к минимуму (по сравнению с другими упомянутыми системами).

24.1 Мультипроцессорный вычислительный комплекс «ЭЛЬБРУС-2» (МКМД с ОП)

В МВК «Эльбрус-2» заложен модульный принцип, включая систему охлаждения и первичной разводки электропитания. Благодаря этому принципу, существенно повышается безотказность МВК «Эльбрус-2», т.к. отказ отдельного модуля (или модулей) не приводит к отказу всего МВК, а некоторому снижению производительности. При соответствующем выборе конфигурации модулей комплекс функционирует практически непрерывно.

Максимальная производительность МВК «Эльбрус-2» составляет 120 млн. оп/с, максимальная ёмкость оперативной памяти – 16 млн. 72-разрядных слов (144 Мбайт), максимальная пропускная способность каналов ввода-вывода – 120 Мбайт/с.

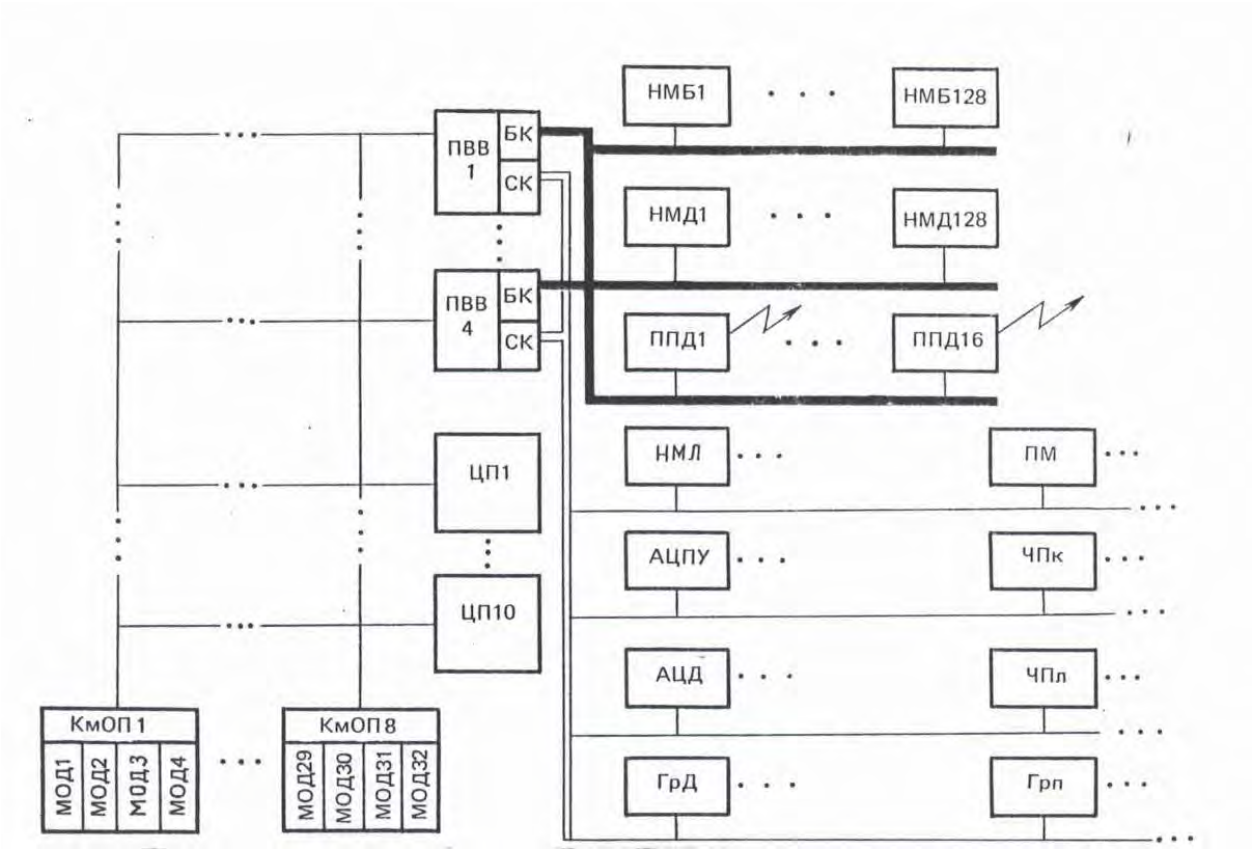


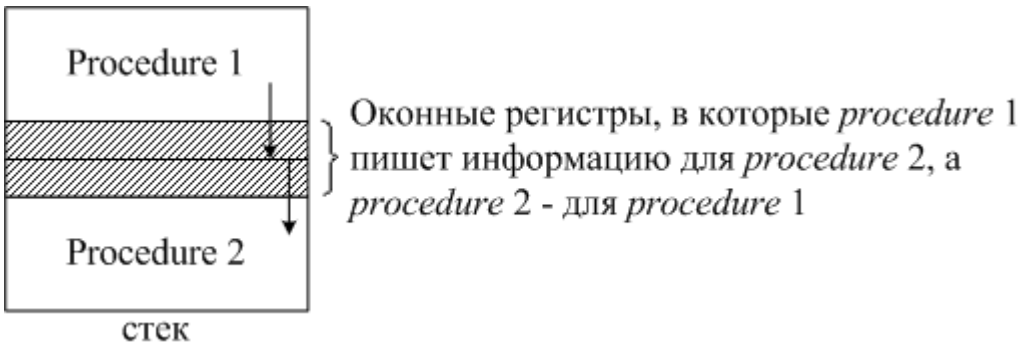
Рис 24.1 Структурная схема МВК «Эльбрус-2»

Особенности:

1. оригинальная система команд, имеющая безадресный формат (стековый механизм);
2. аппаратная поддержка рекурсивного вызова процедур;
3. эффективная работа процессов пользователя с общими данными;
4. динамическое распределение памяти;
5. контекстная защита памяти;
6. разветвлённая система прерываний;

7. аппаратная поддержка параллельной обработки на уровне независимых задач, независимых ветвей задач, на уровне команд.

Оконные регистры. Типичная ситуация, когда одна процедура вызывает другую (вложенную по отношению к вызывающей), чтобы последняя подготовила для неё какие-то данные, способна сильно притормозить вычисления на многопроцессорных ВС, в которых используются векторные команды, за счёт обмена данных между участками памяти. Хотелось бы сделать так, чтобы вызываемая процедура записывала результат в память или регистр, а процедура, которая дожидалась этих данных, немедленно начала бы обрабатывать их. Такие регистры называются оконными.



Обмен данными прост и эффективен!

24.1.1 Центральный процессор

Центральный процессор МВК «Эльбрус-2» – универсальное вычислительное устройство, предназначенное для решения широкого класса научно-технических и информационно-логических задач. В состав ЦП входят устройство управления (УУ), арифметическое устройство (АУ) и сверхоперативное запоминающее устройство (СОЗУ).

Центральный процессор обеспечивает:

1. эффективную трансляцию и выполнение программ за счёт аппаратной реализации наиболее устоявшихся в языках программирования высокого уровня и операционных системах форм описания алгоритмов;
2. мультипрограммный режим работы с аппаратной реализацией защиты данных пользователей и ОС и минимальным временем переключения задач;
3. высокую производительность, основанную как на применении интегральной технологии, так и на параллельной структурной организации и глубоком совмещении выполнения команд;
4. эффективную мультипроцессорную обработку в МВК, имеющем до десяти ЦП и до четырёх ПВВ и т.д.

На рис. 24.2 представлена структурная схема центрального процессора «Эльбрус-2»

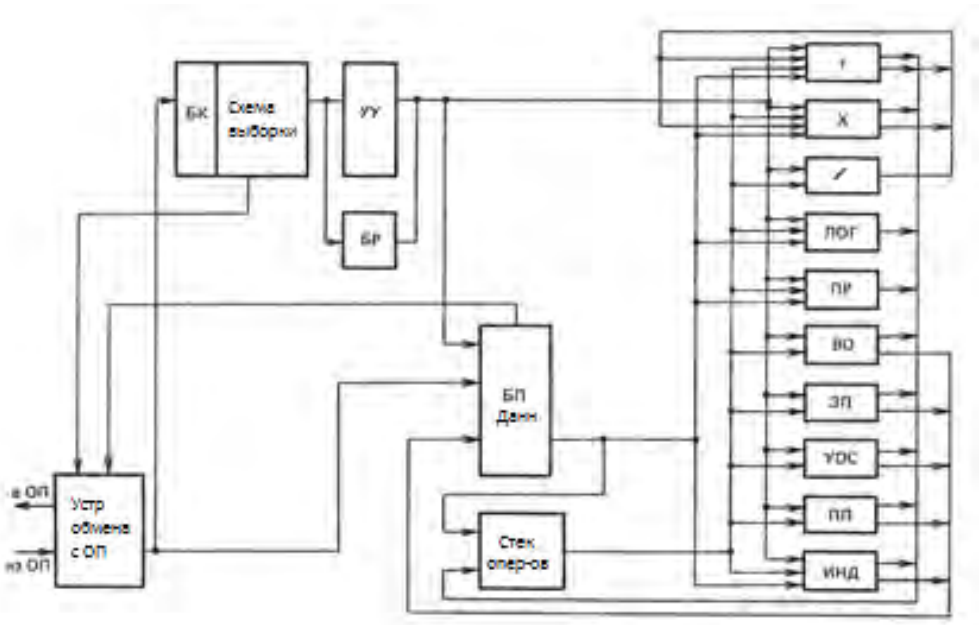


Рис. 24.2 Структурная схема центрального процессора «Эльбрус-2»

Задачи устройства управления(УУ):

1. преобразует команду из внешнего безадресного представления во внутренний регистровый формат. Полем операндов является стек операндов;
2. управляет распределением ячеек памяти стека операндов и буфера в потоке параллельного выполнения команд;
3. распределяет команды по исполнительным устройствам;
4. считывает операнды в исполнительные устройства из стека операндов;
5. выполняет первичное обращение в буферную память команд, в коде которых содержатся адреса;
6. обеспечивает хранение информации о последовательности дешифрации параллельно выполняемых в ЦП команд;
7. обеспечивает восстановление состояния ЦП при прерываниях и отменах условной ветви выполнения программы.

В устройстве управления две пары регистров дешифрации команд; вторая пара регистров дешифрации нужна, если намечается ветвление – записываются команды той ветви, которая наиболее вероятна.

В этой системе выполняются 14 команд: 2 на стадии дешифрации, а 12 – на стадии выполнения в исполнительных устройствах.

БР – базовые регистры предназначены для преобразования относительных адресов из командного потока в абсолютные – всего 32 регистра, каждый из которых описывает определенную область памяти, т.е. содержит начальный адрес области, ее размер. Занесение дескрипторов в БР выполняет исполнительное устройство (ИУ) подпрограмм.

Стек операндов (СтОп) – обычная прямоадресуемая память на 32 двойных слова. Доступ в стек выполняется от 8 ИУ на основе приоритетной схемы. Предназначено для ускорения доступа к данным, размещенным в вершине стека.

Буферная память данных (БП), содержит буфер стека, АЗУ глобалов, буфер массивов.

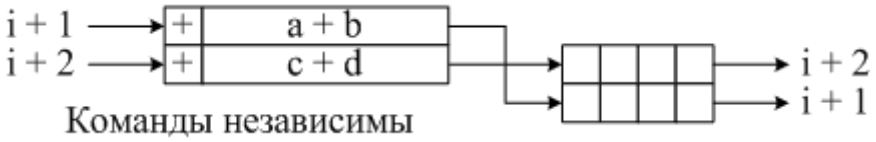
БК – буферная память команд.

СВ – схемы выборки.

УООП – устройство обмена с оперативной памятью.

Исполнительные устройства.

Блоки «сложение» (+) и «умножение» (х) (наиболее часто встречающиеся операции) имеют по две секции, т.е. закачиваются сразу две команды: первая выполняется, вторая уже готова к выполнению. Последовательность выполнения команд может быть нарушена (по сравнению с программной последовательностью).



ПР – ИУ десятичных преобразований – преобразование числа из десятичного представления в битовый набор и наоборот.

Первые пять исполнительных устройств занимаются непосредственно обработкой задачи пользователя, остальные выполняют вспомогательные (служебные) действия.

Сами устройства анализируют код операций и ТЕГ, т.е. дешифрируют команды по логическому назначению

ВО – ИУ вызова операнда в стек.

ЗП – ИУ записи операнда в оперативную память

УОС – устройство обработки строк; предназначенное для обработки символов, цифровой и битовой информации; устройство связано с операциями перемещения массивов в памяти, поиск информации по заданному признаку.

ПП – ИУ подпрограмм; участвует в выполнении операций, связанных со входом одной процедуры, возвратом из них и командами перехода. Устройство выполняет операции считывания/записи в системе прямо адресуемых регистров процессора, операций откачки и подкачки, при регулировании состояния стека операций и операций взаимодействия процессов.

ИНД- индексация; выполняет операции индексации (вычисления адреса элемента массива по заданному дескриптору массива и индексу элемента).

24.1.2 Формат данных ЦП

64р – слово

32р – полуслово (вся управляющая информация, но упаковывается в слово).

Нечисловая информация может быть представлена в битовом формате и в байтовом.

Существует 4 класса информации (По два разряда на класс. Любое слово содержит 2 разряда, по которым ИУ определяет класс):

1. арифметико-логический класс
2. класс адресов (имя, метки процедур и переходов)
3. служебный класс (существует два типа связывающей информации для подпрограмм: вызываемая и вызывающая; служебные списки, которые использует ОС, а также специальные аппаратные средства, которые определяют работу с данным кодом программ физической памяти)
4. класс специальной информации (семафоры):

e := a/b×c – d

сч a,сч b, сч c, ×, сч d, -, /

R1:=a

R2:=b

R3:=c

R2:= b×c

R3:=d

R2:=R2-R3

R1:=R1/R2

Организация группы команд, связанных с записью и считыванию из памяти.

Организация вызова процедур по ссылке осуществляется следующим образом:

Т.к. теги поддерживаются аппаратно, не надо программно анализировать, что мы считаем.

Это одно из главных преимуществ теговой архитектуры.

Типизация данных позволила реализовать алгоритмы (рис. 24.3) передачи параметров по «ссылке» и «процедуре». Если результат считывания – адресная информация, то операция считывания автоматически повторяется по новому адресу, до тех пор, пока не будет считано значение числового типа. Если результат считывания – имя процедуры, то она автоматически запускается и результат ее выполнения рассматривается как считанное значение.

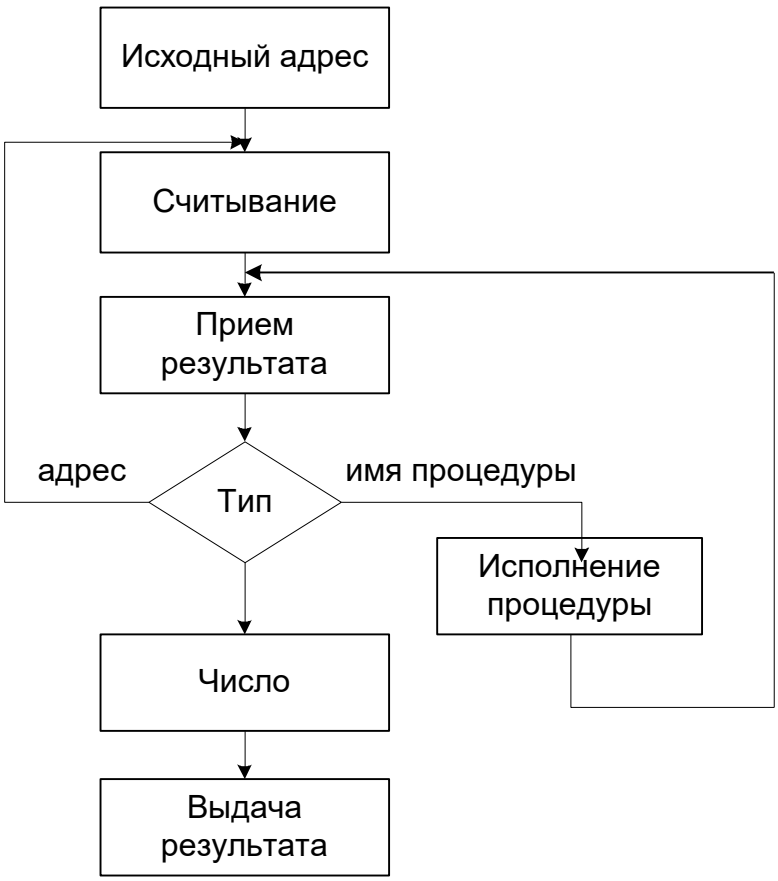


Рис.24.3 Алгоритм передачи параметров

24.1.3 Организация оперативной памяти

Подсистема оперативной памяти является важным компонентом многопроцессорной вычислительной системы и оказывает значительное влияние на её характеристики. Связь центральных процессоров и процессоров ввода-вывода с памятью и между собой, взаимодействие процессоров при параллельной работе, возможные конфликты из-за общих ресурсов системы определяют типы вычислительных систем (сильно- и слабосвязанные, эгалитарные и автократические и т.д.), существенно влияют на производительность, надёжность и гибкость МВК. Коммутационное оборудование может составить значительную часть аппаратуры комплекса.

В МВК «Эльбрус» для связи процессоров с оперативной памятью применена многопортовая-многошинная схема коммутации с топологией соединения по полному графу (каждый с каждым). Такая схема позволяет получить максимальную скорость передачи информации и одновременно обеспечить высокую надёжность и гибкость системы, т.к. наилучшим образом соответствует принципу модульности.

Недостатками такой системы являются большое число линий связи и значительный объём коммутационного оборудования, а также то, что максимальная конфигурация системы ограничена числом портов, имеющихся в коммутаторах процессоров и секций памяти.

Применение многопортовой-многошинной полноперекрёстной схемы коммутации оправдано для высокопроизводительных МВК, состоящих из небольшого числа мощных процессоров и секций памяти большой ёмкости, каким и является МВК «Эльбрус».

Оперативная память МВК «Эльбрус» имеет блочную структуру. Минимальный размер блока – 16 К слов – соответствует объёму памяти 16 Кбит. Блок состоит из 80 микросхем памяти, по одной микросхеме на разряд (слово, хранящееся в памяти, содержит 80 разрядов, включая контрольные разряды кода Хемминга).

Для сокращения числа линий связи, коммутационного и управляющего оборудования, блоки объединяются по иерархической схеме. На рис. 24.4 представлена блочная структура оперативной памяти для одного разряда. Секции ОП (0…7) размещены в отдельных шкафах и имеют линии связи со всеми процессорами. Модули памяти (0…31) имеют собственное коммутационное оборудование и управляющее оборудование. Группы микросхем имеют собственные шины управления и регистровое обрамление.

Для повышения пропускной способности памяти применяется распределение последовательных адресов по разным блокам памяти (*интерливинг*). В МВК «Эльбрус» возможен интерливинг разных уровней: между секциями памяти; между модулями секции; внутри модуля секции памяти, что изображено на рис. 24.4.

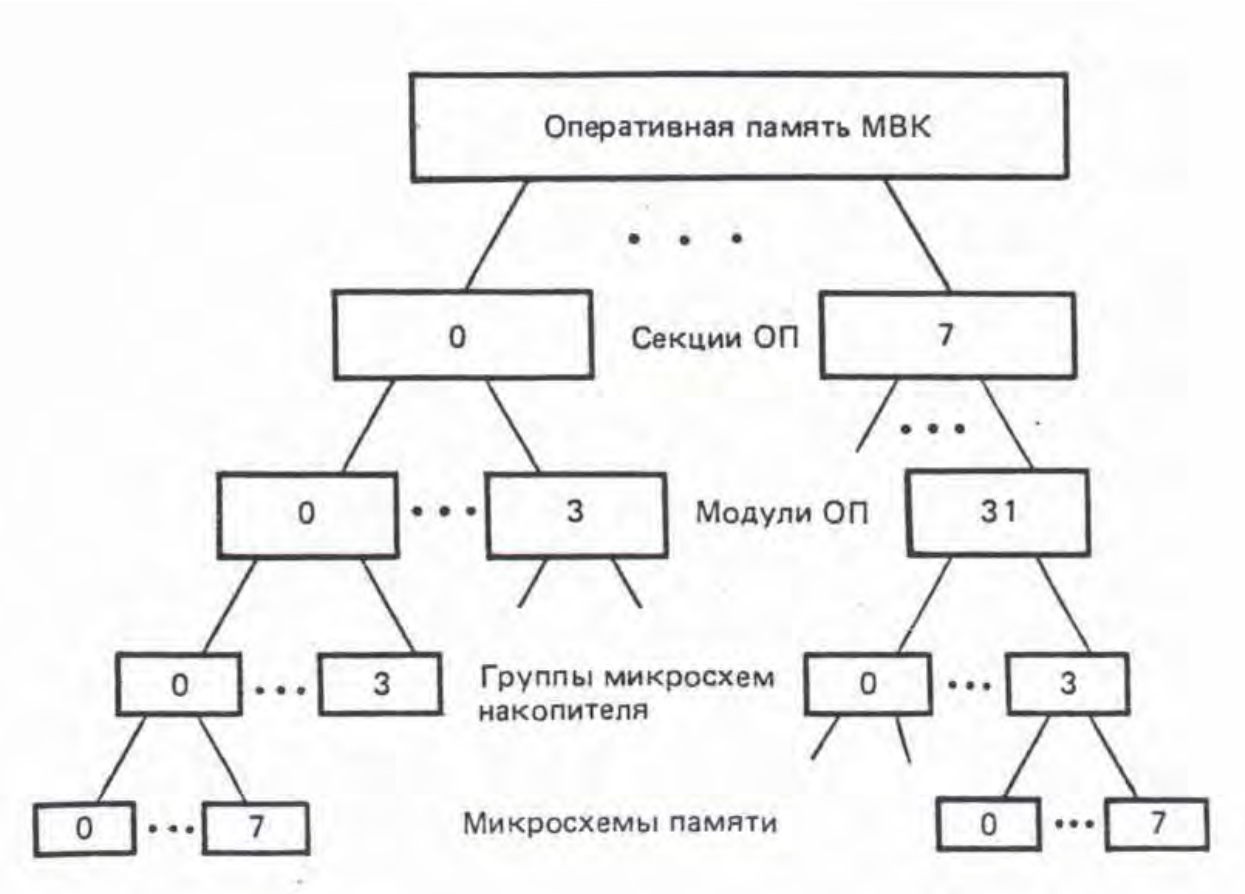


Рис.24.4 Схема доступа к ячейкам памяти

24.2 МВС «Эльбрус-3» (МКМД с ОП)

«Эльбрус-3» строилась как МВС с 16-ю ЦП, работающими в режиме реального времени, пакетном режиме (местный или удаленный), разделение времени, пакетное разделение времени.

При $T_Ц = 10нс$, производительность составляет: 10 млрд. операций в секунду.

5 млрд. операций в секунду – для векторных операций

1,5 млрд. операций в секунду – для скалярных операций

Каждый процессор имеет свою локальную память (основное отличие от «Эльбрус-2»).

Оперативная память состоит из 8 секций по 256 Мб.

Пиковая производительность процессора – 700 млн. оп/с

Для векторных операций – 350 оп/с

Для скалярных операций – 100 млн. оп/с

Структурная схема представлена на рис. 24.5

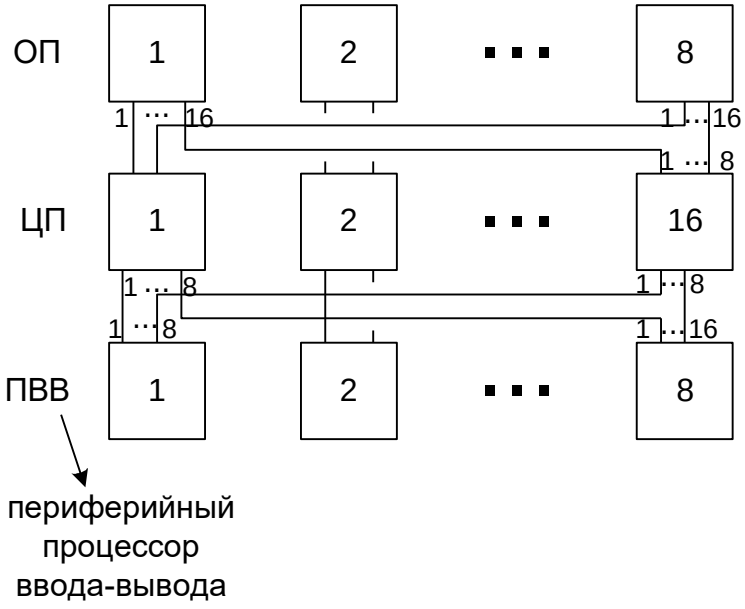


Рис.24.5 Структурная схема «Эльбрус – 3»

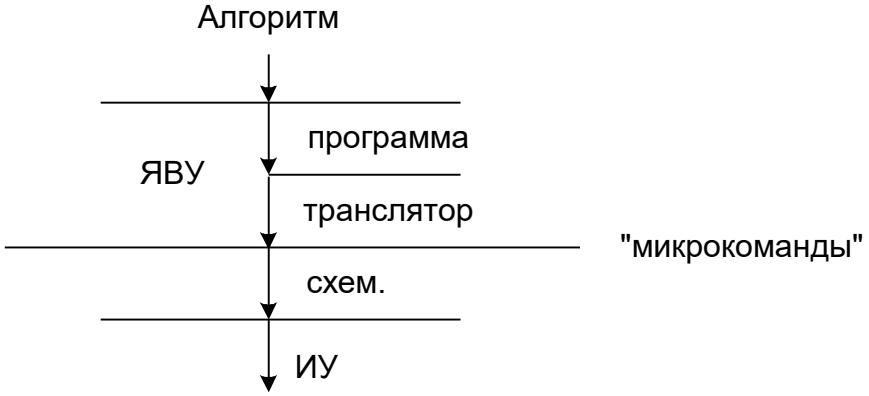
Каждая секция памяти имеет 16 каналов, которые соединяют ее со всеми процессорами.

В каждом процессоре есть свой коммутатор и 8 каналов, соединяющие его со всеми периферийными процессорами.

Основные принципы:

1. теговая архитектура
2. очень длинное командное слово
3. программист работает только с ЯВУ
4. хороший высокооптимизирующий транслятор

На рис.24.6 представлена схема информационной системы «Эльбрус-3».



Целевая функция:
Взять все хорошее у микропрограммного управления и устранить его недостатки.

Рис.24.6 Схема информационной системы «Эльбрус-3»

Достоинства микропрограммного управления:

1. Управление является параллельным.
2. Управление происходит в «реальном» масштабе времени (1 микрооперация – 1 такт)

Недостатки:

1. Управление на мелком уровне
2. Управление осуществляется интерпретационно.

В «Эльбрус-3» управление является параллельным, т.к. в каждой команде (очень длинное командное слово) максимально запускается семь арифметических операций, происходит управление записью результатов в буфер стека, параллельное выполнение условного и безусловного перехода, параллельное обращение по восьми каналам в локальную и (или) глобальную память. Управление происходит в «реальном» масштабе времени – каждый такт на исполнение выдаётся очень длинное командное слово, а это и есть крупный уровень управления. За счёт предварительной компиляции удалось уйти от процедур интепретационного управления.

24.2.1 Основные принципы обработки данных в «Эльбрус-3»

1. Программная совместимость с «Эльбрус – 1,2 »
2. В основе – теговая архитектура, позволяющая проводить динамическую типизацию данных и обеспечивать высокий уровень программирования и исключение Assembler’a.
3. Развитая многопроцессорная архитектура
4. За счет использования очень длинного командного слова (320 разрядов), в каждой команде запускается 7 арифметических операций, результаты которых используются в качестве аргументов для последующих операций (принцип зацепления в чистом виде).
5. Используется конвейер операций
6. ЦУУ очень простое
7. Основная работа по оптимизации и распределению ресурсов возлагается на высокооптимизирующий транслятор, поэтому принцип распараллеливания – статический. (В «Эльбрус-2» - динамический, поэтому ЦУУ очень сложное).
8. Основу эффективной стыковки работы процессоров и ОП составляет функциональное разделение сверхоперативной памяти, по емкости (отличается от «Эльбрус-2»).
    a. Буфер команд - 2К слов (по 72 разряда).
    b. Буфер стека операндов -1К слов (по 72 разряда).
    c. Буфер массивов – 512 слов (по 72 разряда).
    d. Ассоциативная память глобалов -512 слов (по 72 разряда).

Память контролируется кодом Хэмминга.

24.2.2 Структурная схема ЦП «Эльбрус-3»

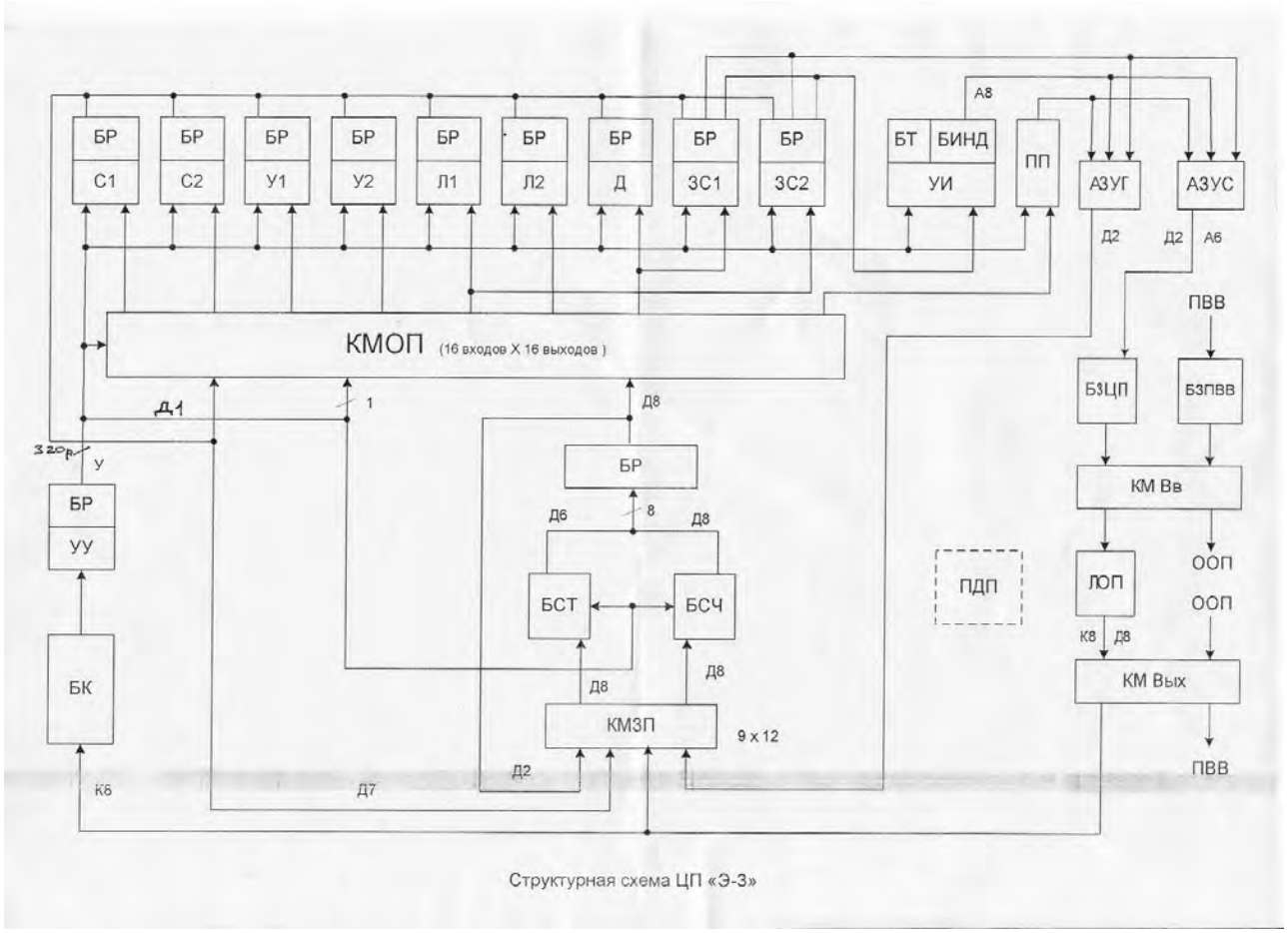


Рис.24.8 Структурная схема центрального процессора Эльбрус-3

А8 – восемь адресов за 1 такт (т. е. параллельно)
Д2 – два слова данных за 1 такт
К8 – восемь команд за 1 такт
Д2 А6 – два слова данных и шесть адресов за 1 такт
У – очень длинное командное слово (320 разрядов)

БТ – буфер трафаретов
БИНД – буфер индексации
УИ – устройство индексации
БСТ – буфер стека активации процедур
БР (которые сверху) – буферы результатов
С1, С2 – сумматоры
У1, У2 – умножители
Л1, Л2 – блоки логических операций
Д – блок деления
ЗС1, ЗС2 – блок записи/считывания
ПДП – планировщик доступа к памяти
ПП – устройство подпрограмм

АЗУГ – АЗУ глобалов
БЗУП – буфер заявок ЦП (центрального процессора)
БЗПВВ – буфер заявок процессора ввода/вывода
КМВВ – коммутатор ввода/вывода
ЛОП – локальная память
КМ Вых – коммутатор выхода
ООП – общая оперативная память
КМОП – коммутатор операндов
КМЗП – коммутатор записи
БК – буфер команд
УУ – центральное устройство управления
БР (которые снизу) – буферные регистры
ПВВ – процессор ввода/вывода
БК и *ПП* – это одна связка; работают асинхронно на фоне статически спланированного конвейера обработки данных
БСЧ и *БСТ* – обеспечивают быстрый доступ операндов на исполнительные устройства (т. е. это типа 2-ух кэш)
БСТ – операнды и параметры процедур
БР – буферы результатов – служат для временного хранения результатов операции; эти результаты могут быть использованы в последующих 1 – 7 тактов после их формирования
АЗУГ – предназначен для сокращения времени доступа к общим данным параллельно выполняемых процессов
АЗУС – выполняет преобразование математических (логических) адресов в физические
КМОП – подает операнды на соответствующие входы

С1 и *С2* выполняют:
- целочисленное сложение
- сложение с плавающей запятой
- арифметические отношения
- преобразование типов и форматов арифметических значений

У1 и *У2* выполняют:
- целочисленное умножение
- умножение с плавающей запятой

Д выполняет:
- деление нацело
- деление с плавающей запятой

Л1 и *Л2* выполняют:
- логические операции
- операции логического отношения
- логическая функция выбора
- операции с полями и тегами
- операции арифметических сдвигов

- операции упаковки и распаковки значений при считывание из памяти и записи в память
- операции формирования шкал
- работа с этими шкалами
- операции работы с байтовыми элементами

ЗС1 и *ЗС2* выполняют:

- формирование адресов для обращения к памяти
- формирование адресов атрибутов и других значений
- формирование вторичных индексов и индексация массивов при обработке их в цикле
- выполнение некоторых системных команд, связанных с подкачкой информации

Примечания:

- Очень длинное командное слово (320 разрядов) – это код, а не сигналы (т. е. не микрокоманда); оно подается на все соответствующие блоки и уже там дешифруется.
- Тег (описание операнда) приклеивается к операнду.
- Связи на Л1 и Д также идут на ЗС2 и ЗС1 соответственно. Если много логических операций, то делений практически не бывает, поэтому работает ЗС1. Ели много делений, то логических операций мало и работает ЗС2.

На рис. 24.9 представлена схема буфера стека операндов (БСТ), который предназначен для хранения локальных и глобальных данных, рабочих переменных и параметров процедур. БСТ представляет собой верхнюю часть стека активации процедур. Шестнадцатидоступная память БСТ получается расслоением адресов на чётные и нечётные.

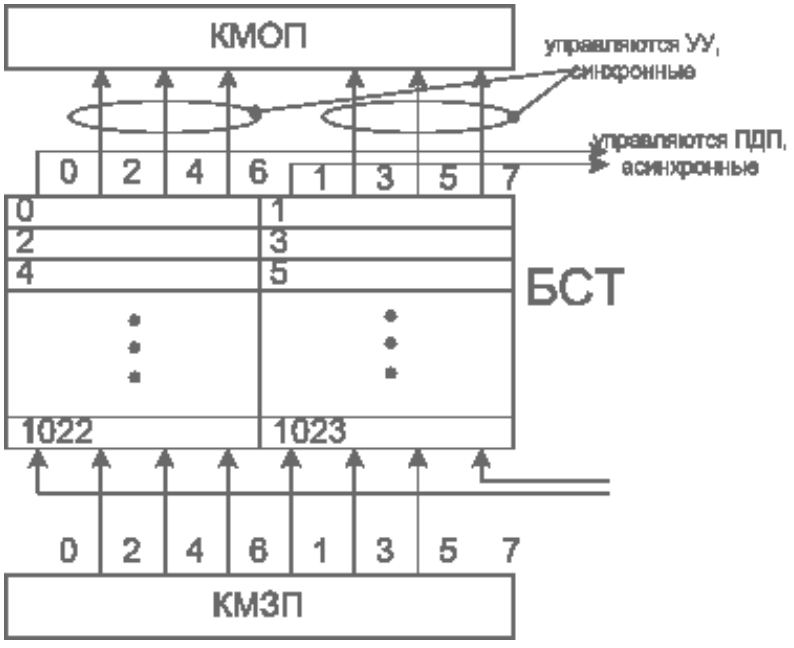


Рис.24.9 Структурная схема БСТ

На рис. 24.10 представлена схема буфера считывания элементов массива (БСЧ), предназначенного для хранения считанных из памяти значений элементов массивов при обработке их в циклах. В каждом такте работы процессора обеспечивается доступ в БСЧ по восьми адресам для считывания значений и восьми адресам для записи значений. Для доступа в БСЧ используется базированный способ адресации. Для этого введены 8 регистров баз (по 4 на каждую половину БСЧ).

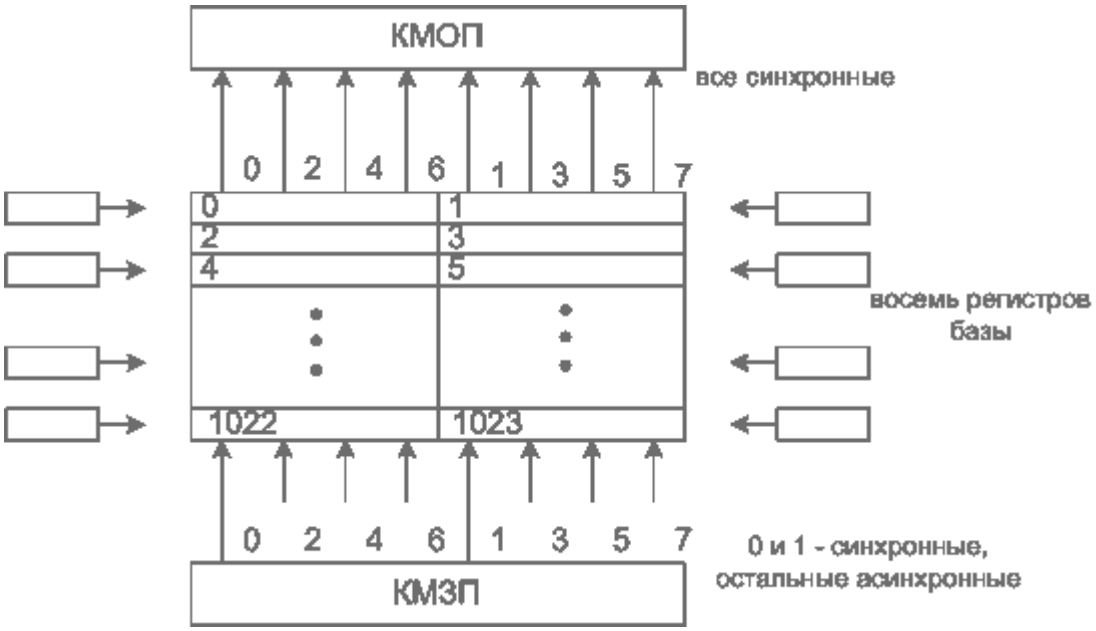


Рис.24.10 Структурная схема БСЧ.

На рис.24.11 представлена структурная схема устройства управления «Эльбрус-3». В УУ реализуется вычисление адреса команды, её выборка и дешифрация. Для работы устройства используется буфер команд (БК).

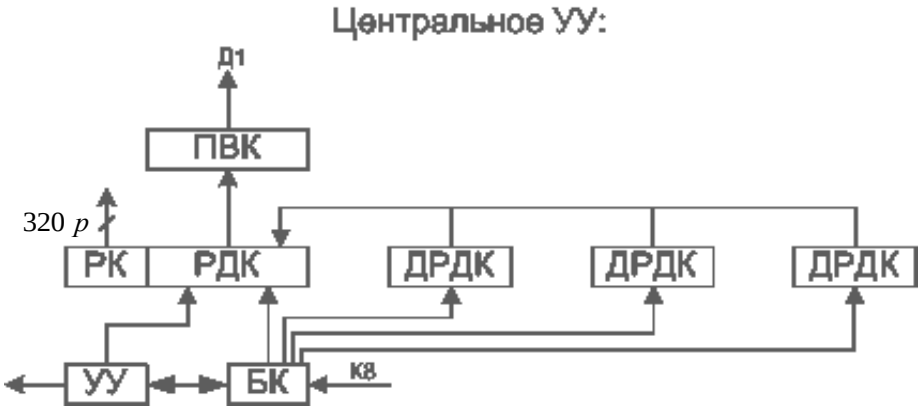


Рис. 24.11 Структурная схема центрального устройства управления

РДК – регистр дешифрации команд
ДРДК – дополнительный РДК
РК – регистр команд

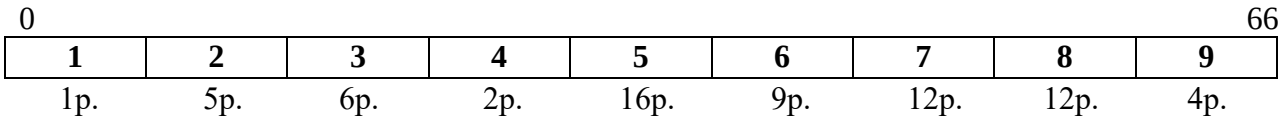
ПВК – протокол выполнения команд, следит за последовательностью выполнения команд
УУ – внутреннее устройство управления
БК – буфер команд, за счёт расслоения обеспечивает считывание 16 и запись до 16 слов в каждом такте работы процессораВ случае ветвления предусмотрена подкачка по 3-м направлениям; за счет этого задержка в выборе направления – всего 1 такт.
Во внутреннее УУ поступает значение соответствующего поля из команды для управления самим центральным УУ.

24.2.3 Структура распакованной команды «Эльбрус-3»

Эльбрус-3 имеет «очень длинное» командное слово:

I – группа управления и литералов;
II – группа результатов выполнения;
III – группа адресов считывания;
IV – группа операций;
V – группа адресов записи.

Группа управления и литералов (I):



Данная группа содержит информацию об управляющих сигналах, которые управляют самим ЦУУ и устройством подпрограмм. Для обработки управляющей информации имеются свои команды управления и константы (литералы).

ПР (1) – формат хранения команды (1- распакованный, 0 - упакованный);
ИСК (2) – определяет номер полуслова, с которого начинается следующая команда в регистре дешифрации команд (емкость регистра – 32 полуслова по 32 бит);
УПРЛ (3) – содержит коды операций команд, которые управляют работой следующих устройств:
1) ЦУУ:
- команды передачи управления;
- команды подготовки переходов;
- пропуска команд;

- команды управления работой циклов;
- команды управления счетчиком циклов;

2) Устройство подпрограмм:
- команды входов/выходов из процедур;
- команды, связанные с переходом по метке;

3) Устройство загрузки полноразрядных литералов;

РДК (4) – 2 разряда, указывающие, в каком регистре дешифрации команд находится текущая команда;

ИК (5) – содержит индекс команды, на которую необходимо передать управление (№ полуслова от начало сегмента программного кода) при условных/безусловных переходах;

СД (6) – поле спусковой функции. Содержит кодировку спусковой функции, значение которой определяется значениями управляющих признаков, поступающих с 9-ти синхронных устройств, определяющих, какие управляющие сигналы будут влиять на условные переходы;

7 и 8 – 12-разрядные литералы, поступающие на вход тех исполнительных устройств, которые заняты преобразованием управляющей информации;

УИЦ (9) – предназначено для управления индексациями в цикле. Используется для запуска вычисления адресов элементов массивов, а также считывания и записи значений в эти элементы.

Группа результатов выполнения (II):

67 87

РС1	РС2	РУ1	РУ2	РД/ЗС	РЛ/ЗС	РЛ

В этой группе имеется 7 полей по числу одновременно запускаемых исполнительных устройств.

РС1, РС2 – регистры сумматоров 1 и 2;
РУ1, РУ2 – регистры умножителей 1 и 2;
РД/ЗС – регистр делителя либо записи/считывания;
РЛ/ЗС – регистр логики либо записи/считывания;
РЛ – регистр логики.

С помощью указанных регистров реализуется принцип зацепления операндов.

Группа адресов считывания (III):

Содержит коды адреса считывания, которые необходимо взять либо из буфера стека операндов, либо из буфера массивов.

88 151

A0	A1	A2	A3	A4	A5	A6	A7
8р.	8р.	8р.	8р.	8р.	8р.	8р.	8р.

С их помощью можно задать:

1) относительный адрес по буферу считывания элементов массивов (буфер – в памяти);
2) относительный адрес по буферу стека процедур;
3) короткий литерал в виде целого без знака.

Т. о. в команде задаются сразу 8 адресов операндов, которые необходимо считать из определенного буфера.

Группа операций (IV):

Содержит 9 полей (для каждого ИУ), каждое из которых делится на 3 части:

152 256

С1	**Ш1С1**	**Ш2С1**	**С2**	**Ш1С2**	**Ш2С2**	**…**
6р.	4р.	4р.	6р.	4р.	4р.	

С1 – код операции для сумматора;
Ш1/Ш2 – адреса выходов КМОП (коммутатора операндов), с которых будут поступать 2 операнда для ИУ.

Группа адресов записи (V):

В этой группе имеется 7 полей (по числу одновременно запускаемых исполнительных устройств):

257 319

АЗС1	**АЗС2**	**АЗУ1**	**АЗУ1**	**…**			
8р.	8р.	8р.	8р.	8р.	8р.	8р.	8р.

АЗС1 – адрес записи результата сумматора 1.
Т. о. 9 разрядов указывают, куда записать результат выполнения операции каждого ИУ (буфер массивов).

Идея упаковки/распаковки в современных микропроцессорных системах не используется (большие накладные расходы), вместо этого используется переменная длина командного слова.

25.Библиографический список

1. **Дерюгин А.А.** Коммутаторы вычислительных систем. – М.: Издательский дом МЭИ, 2008. – 112 с.

2. **Воеводин В.В., Воеводин Вл.В.** Параллельные вычисления, СПб.: БХВ-Петербург, 2002. – 608 с.

3. **Огнев И.В., Борисов В.В.** Ассоциативные среды. – М.: Радио и связь, 2000. – 300 с.

4. **Ладыгин И.И., Логинов А.В., Филатов А.В., Яньков С.Г.** Кластеры на многоядерных процессорах. – М.: Издательский дом МЭИ, 2008. – 112 с.

5. **Каган Б.М.** Электронные вычислительные машины и системы. – М.: Энергоатомиздат, 1985. – 522 с.

6. **Трахтенгерц Э.А.** Программное обеспечение параллельных процессов. М.: Наука, 1987. 272 с.

7. **Озкарахан Э.** Машины баз данных и управление базами данных. – М.: Мир, 1989. – 696 с.

8. **Зюбин В.** Многоядерные процессоры и программирование. «Открытые системы», 07–08, 2005 г.

9. **Многопроцессорные** ЭВМ и методы их проектирования. Бабаян Б.А., Богаров А.В., Волин В.С. и др. – М. : Высш. шк., 1990, – 143 с.

10. **Кузьминский М.** И всё-таки «Эльбрус» рождается!, Computerworld, Россия, 18 ноября 2003 г.

11. **Пахомов С.** Новый процессор Intel Pentium 4 с тактовой частотой 3,06 Ггц. и поддерживаемой технологией Hyper-Threading. «Компьютер пресс», декабрь 2002 г., с. 30–34.

12. **Sun** Fire™ X4100 and X4200 Server Architectures. A technical White Paper. September 2005 SunWIN Token #447327.

13. **Software** Optimization Guide for AMD Athlon™ 64 and AMD Opteron™ Processors., Publication #25112, Revision: 3.0 Issue Date: November 2004

14. **Ладыгин И.И., Калинина Г.А.** Лабораторные работы по курсу «Вычислительные системы» – М.: Изд-во МЭИ, 1999, – 32 с.

26. Список сетевых источников

С1. **Озеров. С, Карабуто А.** Двухъядерные процессоры Intel и AMD: теория, часть 1, 16.06.2005,
http://www.ferra.ru/online/processors/25920.

С2. **Максим Шиша.** Современные внутренние шины – схема приоритетов!
http://www.ferra.ru/online/System/27009

С3. **Озеров С., Карабуто А.** Новые шины. Часть 3. Шина Hyper-Transport – альтернативы нет? опубликовано 23 июля 2004 г.
http://daily.sec.ru/dailypblshow.cfm?rid=17&pid=10969&pos=7&stp=25

С4. **The** InfiniBand Trade Association official website.
http://www.infinibandta.org

С5. **Илья Евсеев**, MPI для начинающих. Учебное пособие.
http://www2.sscc.ru/Links/Litera/il/Default.htm

С6. **В. Шнитман.** Современные высокопроизводительные компьютеры.
http://www.citforum.ru/hardware/svk

С7. **Система** тестов SPEC.
http://www.parallel.ru./computers/benchmarks/spec.html

С8. **Е. Коваленко.** Система Sequent NUMA-Q
http://www.osp.ru/os/1997/02/6.htm