1 История создания вычислительной техники.

Основные этапы архитектуры процессоров:

1. IBM 701, IBM 704 – впервые введена разрядно параллельная память и разрядно параллельная арифметика;
2. IBM-706 – введена арифметика с плавающей точкой;
3. IBM-709 – независимые процессоры ввода/вывода, введено 6 независимых процессоров ввода/вывода;
4. IBM STRETCH – опережающий просмотр команд с тем, чтобы к началу выполнения операций необходимые данные были подкачены в соответствующую память; был решён вопрос о расслоении памяти;
5. ATLAS – введена конвейерная обработка данных;
6. CDC 6600 – введены независимые функциональные устройства для выполнения отдельных операций;
7. CDC 7600 – независимые векторные функционирующие устройства;
8. ILLIAC IV – введены матричные процессоры;
9. CRAY X – изначально спроектированная полностью конвейерная машина, позволяющая осуществлять векторно-конвейерную обработку;

Основные направления, по которым развивались архитектурные решения высокопроизводительной техники:

1. *векторно-конвейерные* компьютеры с различным набором функционирующих устройств, позволяющие оперировать с целыми массивами данных, что существенно ускоряет процесс обработки;
2. *массивно-параллельные* компьютеры с распределённой памятью, их суть: серийно выпускаемые микропроцессоыр объединять посредством некоторых коммуникационных сред, допускается возможность масштабирования. Под масштабированием понимается следующее: включение дополнительных микропроцессоров повышает быстродействие системы в целом;
3. *параллельные компьютеры* с общей памятью; вся оперативная память таких компьюетров является доступной для любого из процессов всей системы, что позволяет быстро осуществлять обмен данными через общие переменные. Главная проблема: синхронизация доступа от нескольких потребителей к одной и той же области памяти. Кроме того, слишком большое число объединяющихся процессоров усложняет управление системой;
4. *компьютеры с кластерной архитектурой*, их суть: несколько компьюетров объединяются в один блок (кластер, группа) с общим управлением входящих в нго компонент. Если вычислительной мощности такого кластера недостаточно, то можно подсоединить новые кластеры, пока не достигнем нужной производительности;
5. *grid-системы*: пользователи обращаются к концетратору (провайдеру), который передаёт запросы на расчётный центр, область с ПО и область с компьютерами, после чего получает от них отклик и результат возвращает пользователям (можно нарисовать схемку, где по центру концентратор, от него двухсторонние стрелочки ко всем четырём блокам + двухсторонняя стрелочка между ПО и компьюетрами)

2 Классификация программного обеспечения

Классификация ПО:

1. *системные программы* (для управления компьютерами и выполнения некоторых вспомогательных функций: создания копий информации, выдача справочной информации, выдача информации об аварийных ситуациях, прогнозирование возможных переходов и т.д.);
   (a) операционная система;
   (b) драйверы;
   (c) утилиты
2. *прикладные программы* (редактирование текста, формирование изображения, выполнение задач из какой-то предметные области, различные игры и т.д.);
   (a) редакторы тексты, графики, звука;
   (b) игры
3. *инструментальные системы* помощь для проектировщикам в разработке нового ПО, средства для создания компиляторов, расширение ОС, оптимизация вычислительного процесса и т.д.
   (a) языки программирования;
   (b) трансляторы;
   (c) базы данных;
   (d) различные вспомогательные системы для выполнения программ, регулирующих аварийное завершение системы, различные риски при создании систем управления и т.п.

3 Два подхода к формированию понятия «архитектура компьютера»

Архитектура как набор взаимодействующих компонент

Архитектура вычислительной системы (ВС) определяет основные функциональные возможности системы, сферу применения, режим работы, характеризует параметры ВС, особенности структуры и т. д.

Вычислительные и логические возможности ВС. Они обусловливаются системой команд (СК), характеризующей гибкость программирования, форматами данных и скоростью выполнения операций, определяющих класс задач, наиболее эффективно решаемых на ВС. К командам управления мы относим команды ввода-вывода данных и команды управления состоянием процессора, памяти и каналов.

Большое влияние на точность выполнения операций оказывают форматы данных.

Аппаратные средства. Простейшая ВС включает модули пяти типов: центральный процессор, основная память, каналы, контроллеры и внешние устройства.

Процессор (УУ + АЛУ + намять) управляет работой системы и обеспечивает вычисления непосредственно по программе. Выполнение машинных команд, команд ввода-вывода (I/O), обращение к памяти, управление состоянием устройств инициализируются или выполняются с помощью процессора.

Основная память предназначена для хранения команд и данных и обеспечивает адресный доступ к ним от процессора.

Каналы – специальные устройства, управляющие обменом данных с внешними устройствами. Каналы инициируют свою работу с помощью процессора и затем переходят в автономный режим работы. Контроллеры ввода-вывода служат для подсоединения внешних устройств (ВнУ)

к каналам и обеспечивают обмен управляющей информацией с внешними устройствами, присвоение приоритетов и выдачу информации о состоянии ВнУ для канала, т. е. это устройства управления ВнУ.

ВнУ служат для ввода-вывода информации с различных носителей.

Память может быть организована как многоуровневая с различным объемом и временем доступа к ней – сверхоперативная (СОЗУ). оперативная (ОП), внешняя (ВнП), так и одноуровневая, виртуальная. Уровни иерархии памяти взаимосвязаны между собой: все данные одного уровня могут быть найдены на более низком уровне.

Программное обеспечение является составной частью архитектуры компьютера и существенно влияет на весь вычислительный процесс.

Операционная система (ОС) управляет ресурсами, разрешает конфликтные ситуации, оптимизирует функционирование системы в целом.

Архитектура как интерфейс между уровнями физической системы

Применительно к ВС термин «архитектура» может быть определен как распределение функций, выполняемых системой, по различным уровням и установление интерфейса между этими уровнями.

Архитектура первого уровня определяет, какие функции по обработке данных решаются системой, а какие передаются внешнему миру: пользователю, оператору ЭВМ. администратору баз данных и т. д. Система взаимодействует с внешним миром через два набора интерфейсов: языки (язык программирования, язык оператора терминала, язык управления заданиями, язык общения с базой данных, язык оператора ЭВМ) и системные программы (программы редактирования, связи, оптимизации, восстановления и обновления информации, интерпретации, управления и т. д., т. е. программы, созданные разработчиком системы). Оба интерфейса должны быть созданы при разработке архитектуры системы.

Уровни, определяемые интерфейсами внутри программного обеспечения, могут быть представлены как архитектура программного обеспечения.

Таким образом, можно сказать, что архитектура компьютера – это абстрактное представление физической системы с точки зрения программиста.

4 Архитектура фон Неймана: принципы, проблемы и способы их решения

Основные характеристики архитектуры фон Неймана:

1. последовательно адресуемая единственная память линейного типа для хранения программ и данных;

2. команды и данные различаются неявным способом лишь при выполнении операций; принимаемые по умолчанию соглашения повзоляют обращаться с командой как с данными, например, для модификации;

3. назначение данных определяется лишь логикой самой программы, т.к. в памяти машины один и тот же набор бит может представлять различные типы данных.

Однако, архитектура фон Неймана плохо ориентирована на выполнение программ на ЯВУ. К сожалению, эти языки в своей структуре имитируют архитектуру фон Неймановского типа. Также возникают проблемы из-за работы компьютера в двоичной системе. В качестве примера можно привести возможность потерь в младшем разряде для десятичного вещественного числа.

Архитектура комьютеров, в основном, совершенствутеся за счёт дополнительных средств.

1. *Области санкционированного доступа.*

Для защиты памяти выделяются так называемые домены, которые представляют собой локальной адресное пространство, определяющее адреса для формирования или использования некоторым набором команд.

Создаваемая структура может быть представлена в виде двух блоков: домена команд и домена данных, которые взаимодействуют только между собой. Команда, не входящая в указанный домент, не сможет обратиться к информации в домене данных. Для усиления защиты можно:

(a) в домене данных определять конкретные точки входа даже для функций из доменна команд;

(b) сегментировать домен данных и к каждому сгементы определить точки входа

Для реализации указанного механизма необходимо создать дополнительное аппаратное обеспечение, поддерживающее области санкционированного доступа;

2. *Одноуровневая память.*

   В программу данные поступают либо через фактические параметры, либо посредством ввода/вывода. В зависимости от типа запоминающего устройства, структура и механизм записи данных могут существенно различаться. Решение данной проблемы лежит в унификации ЗУ. В этом случае файлы станут элементами одноуровневой памяти, причём функции перемещения данных между уровнями ЗУ будут возлагаться на соответствующее ПО и аппаратуру. В отличии от виртуальной памяти, одноуровневая память распространяется на всё запоминающее пространство и не исчезает при завершении работы.

   Плюсы:

   - независимость адресации от принципа адресации памяти;
   - сравнительно низкая стоимость ПО

   Минусы:

   - восстановление памяти;
   - переносимость объектов на системы с традиционной организацией архитектуры;
   - создание встроенного в архитектуру компьютера механизма иерархии ЗУ

3. *Хранение в форме самоопределённых данных.*

   Как правило, информация о типе обрабатываемых данных узнаётся в процессе выполнения команды. Для усовершенствования принципов фон Неймана можно вместе с идентификацией данных указать тип и расположить в данной команде некоторую структуру, которую назовём *тегом*. Кроме того, информацию в теге можно дополнить ещё и сведениями о типе адресации, дополнив формат команды дескриптором. Такой тип адресации называется *теговым форматом*, который позволяет корректно выбирать из всех команд семантичеки соответствующую данному типу.

   Бывает два типа тегов:

   (a) статические (значение определяется до начала выполнения программы и в течение обработки программы не изменяется);

   (b) динамические (определяются по ходу выполнения и постоянно обновляются)

   Теги позволяют обнаруживать некоторые ошибки в программы, защитить информацию от несанкционированного доступа, улучшить отладку программы из-за более информативного дампа памяти, упростить реализацию компилятора и увеличить его скорость исполнения.

   Такая адресация, кроме достоинств, имеет ряд проблем, в частности, требуется значительно увеличить объём памяти для хранения информации.

5 Проектирование архитектуры ВС

Как и всякая сложная задача, разработка архитектуры включает все этапы типового решения:

1. исследования концепции решения;
2. анализ требований, предъявляемых системе;
3. составление спецификаций;
4. изучение известных решений:
   - реинжениринг существующих архитектурных решений;
   - проектирование собственных архитектурных решений
5. разработка функциональной схемы;
6. разработка структурной схемы;
7. автономная отладка компонент;
8. интегральная отладка компонент;
9. аттестация проекта;
10. оценка проекта;
11. установка проекта у заказчика;
12. обучение персонала;
13. сопровождение проекта;
14. поддержка проекта и контроль версий

Для разработки проекта необходимо подобрать коллектив исполнителей, к которым предъявляются следующие требования:

1. профессионализм;
2. коммуникабельность;
3. творческое мышление;
4. желание способствовать функциональному росту коллег;
5. критическое отношение к результату своего труда;
6. постоянное самосовершенствование

Также есть требования к руководителю:

1. формализовать задачу;
2. разбить её на фрагменты;
3. делегировать эти фрагменты компетентным исполнителям

В реальных ситуациях линейная схема не всегда приводит к результату. На некоторых этапах возникает необходимость в связи с различными обстоятельствами венуться на предыдущий шаг с тем, чтобы уточнить, изменить, доработать, адаптировать проект.

6 Типы команд и техника адресации

В машинах с регистрами общего назначения метод (или режим) адресации объектов, с которыми манипулирует команда, может задавать константу, регистр или ячейку памяти.

Метод адресации	Пример команды	Смысл команды	Использование команды
Регистровая	Add R4, R3	R4 = R4 + R3	для записи требуемого значения в регистр
Непосредственная	Add R4, #3	R4 = R4 + 3	для задания констант
Базовая со смещением	Add R4, 100(R1)	R4 = R4 + M(100 + + R1)	для обращения к локальным переменным
Косвенная регистровая	Add R4, (R1)	R4 = R4 + M(R1)	для обращения по указателю к вычисленному адресу
Индексная	Add R3, (R1 + R2)	R3 = R3 + M(R1 + + R2)	полезна при работе с массивами
Прямая	Add R1, (1000)	R1 = R1 + M(1000)	полезна для обращения к статическим данным
Косвенная	Add R1, @(R3)	R1 = R1 + M(M(R3))	если R3 – адрес указателя p, то выбирается значение по этому указателю
Автоинкрментная	Add R1, (R2)+	R1 = R1 + M(R2); R2 = R2 + d	полезна для прохода в цикле по массиву с шагом
Автодекрементная	Add R1, (R2)-	R1 = R1 + M(R2); R2 = R2 - d	
Базовая индексная со смещением и масштабированием	Add R1, 100(R2)[R3]	R1 = R1 + M(100 + + R2) + R3 * d	

Адресация непосредственных данных и литерных констант обычно рассматривается как один из методов адресации памяти (хотя значения данных, к которым в этом случае производятся обращения, являются частью самой команды и обрабатываются в общем потоке команд).

Использование сложных методов адресации позволяет существенно сократить количество команд в программе, но при этом значительно увеличивается сложность аппаратуры. Команды традиционного машинного уровня можно разделить на несколько типов:

1. *арифметические и логические* (целочисленные арифметические и логические операции: сложение, вычитание, логическое сложение, логическое умножение и т.д.);
2. *пересылки данных* (операции загрузки/записи);
3. *управление потоком команд* (безусловные/условные переходы, вызовы процедур и возвраты);
4. *операции с плавающей точкой* (операции сложения, вычитания, умножения, деления над вещественными числами;
5. *десятичные операции* (десятичное сложение, умножение, преобразование форматов и т.д.)

6. *операции над строками* (пересылки, сравнения и поиск строк)

Тип операнда может задаваться либо кодом операции в команде, либо с помощью тега, который хранится вместе с данными и интерпретируется аппаратурой во время обработки данных.

Обычно тип операнда (целый, вещественный, символ) определяет и его размер. Как правило, целые числа представляются в дополнительном коде. Для задания символов компания IBM использует код EBCDIC, другие компании применяют код ASCII. Для представления вещественных чисел с одинарной и двойной точностью придерживаются стандарта IEEE 754. В ряде процессоров применяют двоично-кодированные десятичные числа, которые представляют в упакованном и неупакованном форматах. Упакованный формат предполагает, что для кодирования цифр 0 - 9 используют 4 разряда и две десятичные цифры упаковываются в каждый байт. В неупакованном формате байт содержит одну десятичную цифру, которая обычно изображается в символьном коде ASCII.

7 Иерархия памяти: регистровая, оперативная, главная и вспомогательная.

Память компьютера предназначена для хранения информации. В компьютере имеются два вида памяти: *внутренняя* и *внешняя*.

Внутренняя память расположена в системном блоке. У компьютера есть три вида внутренней памяти: постоянное запоминающее устройство (ПЗУ) на микросхемах памяти только со считыванием — ROM (Real Only Memory), оперативное запоминающее устройство (ОЗУ), так называемая RAM (Random Access Memory), и кэш-память. Микросхемы памяти относятся к разряду ключевых комплектующих ПК: то есть тех, без которых компьютер не может функционировать. Память современных ПК выполнена на интегральных микросхемах.

Информация в ПЗУ сохраняется даже при отключении электропитания. Такой вид памяти используется для хранения внутри компьютера программ системного тестирования, начальной установки конфигурации Setup и ввода/вывода. Совокупность этих программ называется базовой системой ввода/вывода BIOS (Basic Input/Output System).

ОЗУ используется только для временного хранения программ и данных (документов). Чем больший объем оперативной памяти установлен на компьютере, тем быстрее и комфортнее работать на нем пользователю. Оперативное запоминающее устройство строится на микросхемах памяти с произвольным доступом к любой ячейке. Оперативная память бывает либо статической (на триггерах) и называется SRAM (Static RAM), либо динамической (на основе конденсаторных ячеек) DRAM (Dynamic RAM). Быстродействующее запоминающее устройство в компьютерных системах — это память, дающая возможность центральному процессору хранить текущие программы и данные. Процессор компьютера может работать только с данными, которые находятся в оперативной памяти. Данные с диска для обработки считываются в оперативную память. Память ПЗУ— постоянная память компьютера, или постоянное запоминающее устройство — это устройство памяти в виде набора интегральных схем (чипов), часто использующееся в микропроцессорах. Данные записываются в ПЗУ при производстве компьютеров непосредственно на фабрике. Информацию в постоянной памяти нельзя изменять — она инициализируется только для чтения.

Память в персональном компьютере делится на внутреннюю, расположенную на системной плате, и внешнюю. Внутренняя память в свою очередь можно разделить на кэш-память и основную память. Регистровая кэш-память – высокоскоростная память, являющаяся буфером между оперативной памятью и микропроцессором, позволяющая увеличивать скорость выполнения операций. Регистры кэш-памяти недоступны для пользователя, отсюда и название *кэш* (Cache). По принципу записи результатов различают два типа кэш-памяти:

- кэш-память «с обратной записью» – результаты операций прежде, чем записать их в ОП. фиксируются в кэш-памяти, а затем контроллер кэш-памяти самостоятельно перезаписывает эти данные в ОП;

- кэш-память «со сквозной записью» – результаты операций одновременно, параллельно записываются и в кэш-память, и в ОП.

Следует иметь в виду, что может использоваться дополнительная кэш-память (кэш-память 2-го уровня), ёмкость которой может достигать нескольких мегабайтов.

Основная память содержит оперативное (RAM – Random Access Memory – память с произвольным доступом) и постоянное (ROM – Read-Only Memory) запоминающие устройства. Оперативное запоминающее устройство предназначено для хранения информации (программ и данных), непосредственно участвующей в вычислительном процессе на текущем этапе функционирования ПК. ОЗУ – энергозависимая память: при отключении напряжения питания информация, хранящаяся в ней, теряется. Из ПЗУ можно только считывать информацию, запись информации в ПЗУ выполняется вне ЭВМ в лабораторных условиях.

Внешняя память

Устройства внешней памяти или, иначе, внешние запоминающие устройства весьма разнообразны. Их можно классифицировать по целому ряду признаков: по виду носителя, типу конструкции, по принципу записи и считывания информации, методу доступа и т.д. Носитель – материальный объект, способный хранить информацию. В зависимости от типа носителя все ВЗУ можно подразделить на накопители на магнитной ленте и дисковые накопители. Накопители на магнитной ленте, в свою очередь, бывают двух видов: накопители на бобинной магнитной ленте (НБМЛ) и накопители на кассетной магнитной ленте (НКМЛ – стримеры). В ПК используются только стримеры.

8 Организация кэш-памяти

Кэши – один из уровней иерархии памяти. Кэши сравнительно небольшие по объёму, но имеют высокую скорость доступа. В кэш-памяти хранятся команды и данные, которые часто используются и требуют малых временных затрат для доступа к ним. Обычно никакое явное (программное) управление входами кэша невозможно. Данные распределяются и управляются кэшем автоматически. Кэш-память позволяет организовать работу медленной оперативной памяти как быстродействующей, оптимизируя следующие аспекты:

- максимизация коэффициента попадания;
- уменьшение времени доступа;
- уменьшение штрафа промаха;
- уменьшение непроизводительных затрат времени, требуемых для поддержания консистентности (согласованности данных с программой) кэша.

Коэффициент попадания в кэш (в %) определяется отношением количества успешных входов к числу промахов. Кэши первого уровня обычно нуждаются в одном импульсе сигнала, чтобы выбрать запись.

Принципы создания

Эффективность кэширования базируется на свойстве локализации ссылок во времени и локализации в пространстве. Реализуя *принцип локализации во времени*, каждый элемент (команда, данные), к которому обращается процессор, копируется в кэш, где он хранится до очередного востребования. Поскольку обращения к памяти носят не случайный характер, а происходят в соответствии с выполняемой программой, то при считывании данных из памяти с высокой степенью вероятности можно предположить, что в ближайшем будущем опять произойдёт обращение к этим же данным. Реализуя *принцип локализации в пространстве*, наряду с текущим элементом в кэш копируется и несколько близлежащих элементов.

В системах, оснащённых кэш-памятью, каждый запрос к оперативной памяти выполняется следующим образом:

1. просматривается содержимое кэш-памяти с целью определения, не находятся ли нужные данные в кэш-памяти;

2. при возникновении промаха контроллер кэш-памяти выбирает подлежащий замещению блок.

Как правило, используются две стратегии:

- случайная (блок выбирается случайно);
- LRU (Least-Recently-Used) - заменяется блок, не использовавшийся дольше остальных)

Если некоторый блок основной памяти может располагаться в любом месте кэш-памяти, то кэш называется *полностью ассоциативным*. Если блок основной памяти может располагаться на ограниченном множестве мест в кэш-памяти, то кэш называется *множественно-ассоциативным*. Обычно множество представляет группу из нескольких блоков в кэше. Если множество состоит из n блоков, то такое размещение называется множественно-ассоциативным с n каналами.

Принципы размещения блоков определяют основные типы организации кэш-памяти. Если каждый блок основной памяти имеет только одно фиксированное место, на котором он может появится в кэш-памяти, то такая кэщ-память называется *кэшем с прямым отображением*. Для записи в кэш-память имеется два способа:

- сквозная запись (write through) – информация записывается сразу в блок кэша и в блок более низкого уровня памяти;
- запись с обратным копированием (write back, copy back) - информация записывается только в блок кэш-памяти

Модифицированный блок кэш-памяти записывается в основную память только в случае его замещения. Для сокращения частоты копирования блоков при замещении с каждым блоком связывается бит модификации (dirty bit), показывающий, был ли изменён данный блок. Когда процесс ожидает завершения обращения к памяти при выполнении сквозной записи, то говорят, что он приостанавливается для записи (write stall). Для минимизации приостановок используется буфер записи (write buffer), который позволяет процессору продолжить выполнение других команд во время обновления содержимого памяти. Следует отметить, что приостановки могут возникать и при наличии буфера записи.

9 Концепция виртуальной памяти и преобразование адресов

Каждая часть среды компьютера имеет собственное обозначение: ячейка – адрес, периферийное устройство – номер и т. д. В простейших компьютерах собственные обозначения указывается непосредственно в программе. В более сложных компьютерах программа отделена от среды «аппаратом преобразования собственных обозначений». Рассмотрим один элемент среды – память. Аппарат преобразования адреса (АПА) не находится под прямым управлением программы и связь с ним осуществляется только через процедуры, работающие в управляющем режиме. Если программисту безразлично существование АПА, то он работает с набором ячеек и периферийных устройств, образующих «виртуальную (математическую, мнимую) среду». Почти всегда виртуальная среда есть переупорядоченное подмножество реальной среды. Каждому виртуальному элементу соответствует реальный элемент, но обратное не всегда верно. Рассмотрим один из элементов виртуальной среды – виртуальную память (ВП).

Задачи, решаемые виртуальной памятью

Виртуальный адрес – адрес, по которому ссылаются на ячейку виртуальной памяти. Область виртуальных адресов – это множество всех виртуальных адресов. Использование виртуальной адресации обусловливается следующими обстоятельствами.

Однородность области адресов.

Представим себе реальный компьютер без виртуальной памяти. Пусть на нем выполняется параллельно несколько процессов. У каждого процесса будет отдельная локальная среда и каким-то образом распределяемые элементы общей среды. Программисту требуется заранее знать, к каким конкретно частям общей среды его процедура может обращаться. Это затруднительно для пользователей ЭВМ, составляющих свои собственные программы. Отвести наперед фиксированную область среды для каждого процесса невозможно, ибо положение каждой конкретной программы определяется положением всех других программ. При виртуальной адресации каждый процесс может выполняться в памяти начиная с фиксированной ячейки, имеющей необходимые размеры области ЗУ. Автору безразлично, в каком участке памяти выполняется его программа, так как каждое обращение к виртуальной памяти во время выполнения посредством АПА преобразуется в реальное обращение.

Защита памяти.

Общеизвестно, что основная цель защиты памяти состоит в том, чтобы не дать возможности некорректному процессу испортить часть среды, относящуюся к другому процессу. Особенно это важно при защите сред управляющих процедур. Виртуальная адресация здесь используется следующим образом: при каждой ссылке процессом на память проверяется, принадлежит ли она к области виртуальных адресов, отведенных для данного процесса.

Изменение структуры памяти.

При проектировании больших программ структура памяти машины с малой ОП явно усложняет проектируемую программу. Применение виртуальной адресации позволяет преобразовать память на разных ступенях иерархии в «одноуровневую память» с одинаковым доступом ко всем элементам и ее отображение на реальную память.

Страничная организация памяти.

Отображение виртуальной памяти в реальную обычно осуществляется с помощью страничной организации памяти. Виртуальную намять в системе со страничной организацией памяти делят на ряд «блоков» фиксированной длины, равной $2k$, где k целое натуральное число. Так как первая ячейка блока $N+1$ примыкает к последней ячейке блока N, то программисту факт разбиения ВП на блоки учитывать не требуется. Оперативная память компьютера делится на «страницы», а вспомогательная – на «сегменты» такого же размера. Виртуальную память пользователя можно разделить на три типа: «активные» блоки, которые содержат программу и данные, используемые в текущий момент; «пассивные» блоки, содержащие программу и данные, которые будут использоваться при выполнении программы; «мнимые» блоки, к которым не обращаются на протяжении выполнения программы.

10 Флеш-память

Флеш-память (ФП) – это современный тип электронно-перепрограемируемого ПЗУ. Эта память стирается и записывается блоками. Это особый вид памяти, который используется, как правило, для мобильных устройств благодаря своим параметрам. Она допускает перезапись данных (от 10^4 до 10^6 раз), которые в ней хранятся, и не требует для хранения дополнительной энергии (только для записи). Отсутствие механических движущихся частей очень сильно снижает потребление энергии, что является одним из основных преимуществ флэш-памяти.

Время хранения информации на флэш-памяти варируется от 20 до 100 лет, при этом механические нагрузки, которые она может выдерживать, превышают предельно допустимые для обычных жёстких дисков в 5-10 раз.

ФП очень компактна: размер составляет от 20 до 40 мм по длине или ширине при толщине до 3мм.

Основные типы ФП:

1. CompactFlash (CF) – 42x36x4 мм, два типа CF I и CFII, самый универсальный стандарт;
2. SmartMedia – дешёвая и ультратонкая (толщина $\approx 3/4$ миллиметра), простая конструкция, невысокая защищённость от случайного стирания;
3. Multimedia Card (MMC) – миниатюрность, низкое энергопотребление, низкая скорость чтения/записи;
4. SecureDigital (SD) – более высокая скорость чтения и более высокая ёмкость, чем у MMC; карты MMC можно использовать через разъём SD;
5. MemoryStick – скорость чтения/записи сравнимы с SD, невысокая ёмкость;

11 Оперативная память. ПЗУ. Структура записи данных.

Оперативное запоминающе устройство бывает двух типов: статическое и динамическое.

В статическом ОЗУ(СОЗУ) информация сохраняется до тех пор, пока к нему подаётся питание. Время доступа к информации – несколько наносекунд, что позволяет использовать статическое ОЗУ в качестве кэш-памяти.

Динамическое ОЗУ(ДОЗУ) хранят в себе информацию через состояния конденсаторов, которые нужно перезаряжать, чтобы не потерять информацию. Динамическое ОЗУ при большом по сравнению со статическом ОЗУ временем доступа к информации (десятки наносекунд) имеет большой объём.

ДОЗУ имеет очень высокую плотность записи, поэтому для использования достоинств обоих типов ОЗУ, используют следующее сочетание: основная память реализуется на ДОЗУ, а кэш-память – на СОЗУ.

Два типа ДОЗУ:

1. Fast Page Mode (FPM) – быстрый постраничный режим, поиск данных вначале по адресу строки, затем по адресу стоблца;
2. Extended Date Output (EDO) – обеспечивает обращение к памяти до завершения предыдущего обращения, что увеличивает пропускную способность вывода

Потребность в сохранение данных даже при выключенном питании привело к созданию ПЗУ – *постоянно запоминающих устройств*. Запись в ПЗУ производится во время его изготовления.

Программируемые ПЗУ позволяют осуществлять их программирование в условиях эксплуатации. Также существует и стираемое программируемое ПЗУ, из которого можно удалять информацию в процессе использования с помощь ультрафиолетового света.

12 Управление памятью.

Управление память, как правило, требует решения двух задач:

- *распределение памяти* между процессами;
- *защита памяти* от несанкционированного доступа

Чтобы решить эти задачи, рассмотрим различные пространства адресов.

Логическое пространство адресов представляет собой множество числовых значений, которые могут быть использованы для доступа к памяти в машинных командах. Мощность множества зависит от конкретной архитектуры ВС. Логический адрес часто представляется в формате

«базовый адрес + смещение», где в качестве базового адреса может быть, к примеру, значение сегментного регистра.

Физическое пространство адресов ограничивается лишь объёмом реальной памяти компьютера.

При преобразовании логических адресов в физические возможны два случая:

1. пространство физических адресов *больше* пространства логических;

   Для эффективного использования физической памяти компьютера используют специальные механизмы расширения памяти, например, реализация EMS (Extended Memory System).

2. пространство физических адресов *меньше* пространства логических;

   В этом случае используют концепцию виртуальной памяти. На диске создаётся файл подкачки, который разбивается на страницы, причём подлежащая распределению оперативная память также разбивается на страницы того же размера. Операционная системе заводит таблицу соответствия логических и физических страниц. При обращении к логическому адресу выполняется проверка, есть ли эта страница в оперативной памяти. При её отсутствии происходит загрузка страницы с возможным вытеснением на диск ранее загруженной страницы.

В соответствии с механизмом использования памяти различают следующие *типы исполняемых модулей*:

- простые (линейные);

  Каждая процедура получает свой блок логических адресов, которые не пересекает с никаким другим блоком адресов другой процедуры.

- оверлейные (с перекрытием);

  Выделяются несколько модулй, которые не обращаются друг к другу, но имеющие общую точку перекрытия, с которой происходит загрузка в память всех перекрывающихся в этой точке модулей.

- динамические

  В отличии от простых и оверлейных схемы, динамическая схема позволяет определить информацию о подпрограммах, которые могут быть затребованы, во время работы соответствующего процесса.

13 Модели консистентности памяти

Модель консистентности представляет собой некоторый договор между программами и памятью, в котором указывается, что при соблюдении программами определенных правил работа модуля памяти будет корректной, если же требования к программе будут нарушены, то память не гарантирует правильность выполнения операций чтения/записи.

Основные модели консистентности:

- Строгая консистентность;

  Операция "чтение ячейки памяти с адресом x"должна возвращать значение, записанное самой последней операцией "запись"с адресом x. В связи с этим должно присутствовать единое подобие абсолютного время.

- Последовательная консистентность;

  Результат любого выполнения такой же, как в случае, если бы операции всех процессоров были выполнены в некотором последовательном порядке, и операции каждого отдельного процессора появлялись в этой последовательности в порядке определенном его программой.

- Причинная консистентность;

  Не требуется, чтобы все процессы видели одну и ту же последовательность записей в память, проводя различие между потенциально-зависимыми (запись одной может зависеть от результата чтения другой ячейки) и потенциально-независимыми (параллельными) операциями записи.

- PRAM консистентность;

- Процессорная консистентность;

- Слабая консистентность;

  Модель консистентности, определяемая тремя правилами:

  1. доступ к синхронизационным переменным определяется моделью последовательной консистентности;
  2. доступ к синхронизационным переменным запрещен, пока не выполнены все предыдущие операции записи;
  3. доступ к данным по записи или чтению запрещен, пока не выполнены все предыдущие обращения к синхронизационным переменным для данного (локального) процессора.

- Консистентность в конечном счете;

  Консистентная в конечном счете система гарантирует, что, в отсутствии изменений данных, в конечном счете все запросы будут возвращать последнее обновленное значение.

- Консистентность по выходу;

- Консистентность по входу.

Отдельно выделим модель линеаризуемости программы, в которой вместо операций чтения и записи рассматриваются операции над объектами (например функции, процедуры), а состояние памяти в данной модели – это состояния объектов. Эта модель используется для систем с объектной организацией общей памяти. В отличие от всех остальных систем, здесь программы не могут напрямую использовать общие переменные (состояние объектов), а только через специальные функции-методы (операции). Для этих систем линеаризуемость совпадает со строгой консистентностью.

14 Графические процессоры. Особенности вычислений. CUDA

Графический процессор – отдельное устройство персонального компьютера или игровой приставки, выполняющее графический рендеринг. Современные графические процессоры очень эффективно обрабатывают и отображают компьютерную графику, благодаря специализированной конвейерной архитектуре они намного эффективнее в обработке графической информации, чем типичный центральный процессор.

Графический процессор в современных видеоадаптерах применяется в качестве ускорителя трёхмерной графики, однако его можно использовать в некоторых случаях и для вычислений (GPGPU).

Отличительными особенностями по сравнению с ЦП являются:

- архитектура, максимально нацеленная на увеличение скорости расчёта текстур и сложных графических объектов;

- ограниченный набор команд.

Высокая вычислительная мощность GPU объясняется особенностями архитектуры. Если современные CPU содержат несколько ядер, графический процессор изначально создавался как многоядерная структура, в которой количество ядер измеряется сотнями. Разница в архитектуре обусловливает и разницу в принципах работы. Если архитектура CPU предполагает последовательную обработку информации, то GPU исторически предназначался для обработки компьютерной графики, поэтому рассчитан на массивно параллельные вычисления.[1] Каждая из этих двух архитектур имеет свои достоинства. CPU лучше работает с последовательными задачами. При большом объеме обрабатываемой информации очевидное преимущество имеет GPU. Условие только одно – в задаче должен наблюдаться параллелизм.

CUDA – программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia.

CUDA SDK позволяет программистам, используя упрощённый диалект C, реализовывать алгоритмы, выполнимые на графических процессорах Nvidia, и включать специальные функции в текст программы на C. Архитектура CUDA позволяет разработчику по своему усмотрению управлять памятью графического ускорителя и организовывать доступ к его набору инструкций.

15 Компьютеры в режиме управления технологическим процессом

Для автоматизации управления сложным производственным или технологическим процессом в контур управления включают компьютер, т. н. управляющую вычислительную машину (УВМ).

Наиболее часто в качестве управляющей ЭВМ используют цифровую ЭВМ благодаря следующим ее качествам:

- наличию больших и высоконадежных ЗУ различного типа;
- возможности решения на них сложных вычислительных и логических задач;
- гибкости (за счет программы);
- надежности и быстродействию

В общем случае система автоматического управления с УВМ определяет собой замкнутый контур:

1. $x_1, x_2, \ldots, x_n$ – измеряемые параметры:
   - нерегулируемые (характеристики исходного продукта);
   - выходные параметры, характеризующие качество продукта;
   - выходные параметры, по которым определяется эффективность производственных процессов или ограничения, наложенные на условия его протекания;
2. $y_1, y_2, \ldots, y_n$ – регулируемые параметры, которые могут изменяться исполнительными механизмами (ИМ) – регуляторами и оператором;
3. $f_1, f_2, \ldots, f_n$ – нерегулируемые и неизмеряемые параметры.

На вход УВМ от датчика Д идет информация о текущем значении параметров $x_1, x_2, \ldots, x_n$. Согласно алгоритму управления УВМ определяет величину управляющих воздействий $U_1, \ldots, U_n$, которые необходимо приложить к ИМ для изменения регулируемых параметров $y_1, y_2, \ldots, y_n$ с тем, чтобы управляющий процесс протекал оптимально. Измерительные датчики вырабатывают непрерывный сигнал (напряжение, ток, угол поворота), а ЦВМ работает в дискретной форме, поэтому 2 раза идет преобразование из непрерывной формы в дискретную (Н/Д) и наоборот (Д/Н).

Для уменьшения оборудования преобразователи Н/Д и Д/Н выполнены многоканальными. Посредством коммутатора преобразователь поочередно подключается к каждому датчику и осуществляется преобразование Н/Д. Поступившее управляющее воздействие U сохраняется в цепи до выработки следующего управляющего воздействия УВМ.

Теперь наибольшее распространение получил *синхронный принцип связи* УВМ с объектом, при котором процесс управления разбивается на циклы одинаковой продолжительности при помощи тактирующих импульсов, поступающих от электронных часов. Цикл начинается с приходом тактирующего импульса на устройство прерывания. В начале каждого цикла проводятся последовательный опрос и преобразование в цифровую форму сигналов датчиков. После выработки управляющих воздействий U_i и преобразовании их в непрерывную форму УВМ останавливается до прихода нового тактирующею импульса или выполняет какую-нибудь вспомогательную работу.

Для установления более тесной связи объекта с УВМ используют *асинхронный принцип связи* с объектом. В некоторых системах применяют комбинированный способ – синхронизацию «плюс» датчики аварийного состояния, переводящие УВМ на режим аварийной работы. В замкнутом контуре УВМ прямо воздействует па ИМ, непосредственно управляя производственным процессом. Это режим прямого цифрового управления.

УВМ в разомкнутой цепи используется:

- в системах автоматического программного управления;

  Уменьшается объем первоначальных входных данных в УВМ для расчета оптимальных настроек регуляторов. Детальный расчет программы управления с заданной точностью производится самим вычислительным устройством, которое вырабатывает необходимые управляющие воздействия в соответствии с этой программой.

- в системах, где УВМ выполняет функции советчика.

  УВМ обрабатывает измерительную информацию е объекта и рассчитывает управляющие воздействия для оптимизации процесса. Эта информация служит рекомендацией оператору, управляющему процессом.

16 CISC-и RISC-архитектуры.

CISC-архитектура

Двумя основными архитектурами набора команд, используемыми компьютерной промышленностью на современном этапе развития вычислительной техники, являются архитектуры CISC (Complete Instruction Set Computer) и RISC (Reduced Instruction Set Computer).

Простота архитектуры RISC-процсссора обеспечивает его компактность, практическое отсутствие проблем с охлаждением кристалла, чего нет в CISC-процессорах. Формирование архитектуры CISC произошло из-за перенесения обработки данных с программного уровня на системный, т.к. предполагалось, что производительность CISC-процессора будет повышаться за счёт упрощения компилятора и минимизации модуля исполнения. На сегодняшний день CISC-процессоры занимают на компьютерном рынке сектор персональных компьютеров, RISC-процессоры являются лучшими в секторе высокопроизводительных серверов и рабочих станций.

Одним из важных преимуществ RISC-архитектуры является высокая скорость арифметических вычислений. Высокая скорость выполнения арифметических операций и высокая точность вычислений даёт преимущество RISC-процессорам в быстродействии над CISC-процессорами.

Другой особенностью RISC-процессоров является комплекс средств, обеспечивающих безостановочную работу арифметических устройств: механизм динамического прогнозирования ветвлений, большое количество оперативных регистров, многоуровневая встроенная кэш-память. Организация регистровой структуры – основное достоинство и основная проблема RISC. Практически любая реализация RISC-архитектуры использует трехместные операции обработки, в которых результат и два операнда имеют самостоятельную адресацию – R1 : = R2, R3. Это позволяет без существенных затрат времени выбрать операнды из адресуемых оперативных регистров и записать в регистр результат операции. Трехместные операции позволяют компилятору более гибко работать по сравнению с двухместными операциями формата «регистр – память» архитектуры CISC. В сочетании с быстродействующей арифметикой RISC-операции «регистр – регистр» становятся очень мощным средством повышения производительности процессора.

Опора на регистры также является и недостатком RISC-архитектуры. Проблема в том, что в процессе выполнения задачи RISC-система неоднократно вынуждена обновлять содержимое регистров процессора за минимальное время, иначе может случиться длительный простой системы. CISC-архитектура не обладает таким изъяном, поскольку модификация регистров может происходить на фоне обработки команд формата «память – память»

CISC-архитектура	*RISC-архитектура*
Многобайтовые команды	Однобайтовые команды
Малое количество регистров	Большое количество регистров
Сложные команды	Простые команды
Одна или менее команд за один цикл процессора	Несколько команд за один цикл процессора
Традиционно одно исполнительное устройство	Несколько исполнительных устройств

RISC-архитектура

Суть ее состоит в выделении наиболее употребительных операций и создании архитектуры, приспособленной для их быстрой реализации. Это позволило в условиях ограниченных ресурсов разработать компьютеры с высокой пропускной способностью.

Четыре основных принципа RISC-архитектуры:

- каждая команда независимо от ее типа выполняется за один машинный цикл, длительность которого должна быть максимально короткой;
- все команды должны иметь одинаковую длину и использовать минимум адресных форматов, что резко упрощает логику центрального управления процессором;
- обращение к памяти происходит только при выполнении операций записи и чтения, вся обработка данных осуществляется исключительно в регистровой структуре процессора;
- система команд должна обеспечивать поддержку языка высокого уровня.

17 Компьютеры со стековой архитектурой.

Стек – это область оперативной памяти, которая используется для временного хранения данных и операций. Доступ к элементам стека осуществляется по принципу LIFO (Last In First Out). Кроме того, доступ к элементам стека осуществляется только через его вершину, т.е. пользователю имеет доступ лишь к последнему элементу, загруженному в стек.

При выполнении различных вычислительных процедур процессор использует либо новые операнды, до сих пор не выбиравшиеся из памяти компьютера, либо операнды, употреблявшиеся в предыдущих операциях. Стековая память представляет собой набор из n регистров, каждый из которых способен хранить одно машинное слово. Одноименные разряды регистров $P_1, P_2, \ldots, P_n$ соединены между собой с помощью цепей сдвига.

Информация в стеке может продвигаться между регистрами вверх и вниз.

- Движение вниз: $(P_1) \longleftarrow P_2, (P_2) \longleftarrow P_3, \ldots$, а P_1 заполняется данными из главной памяти;
- Движение вверх: $(P_n) \longrightarrow P_{n-1}, (P_{n-1}) \longrightarrow P_{n-2}, \ldots$, а P_n заполняется нулями.

Регистры P_1 и P_2 связаны с АДУ, образуя два операнда для выполнения операции, результат которой записывается в P_1. Следовательно, АЛУ выполняет операцию $P_1 \circledR P_2 \longrightarrow P_1$

Одновременно с выполнением арифметической операции (АО) осуществляется продвижение операндов вверх, не затрагивая P_1.

Таким образом, АО используют подразумеваемые адреса, что уменьшает длину команды. В принципе, в команде достаточно иметь только поле, определяющее код операции. Поэтому компьютеры со стековой памятью называют безадресными. В то же время команды, осуществляющие вызов или запоминание информации из главной памяти, требуют указания адреса операнда. Поэтому в ЭВМ со стековой памятью используются команды переменной длины. Команды располагаются в памяти в виде непрерывного массива слогов независимо от границ ячеек памяти. Это позволяет за один цикл обращения к памяти вызвать несколько команд. Для эффективного использования возможностей такой памяти в ЭВМ вводятся спецкоманды:

- дублирование – $(P_1) \longrightarrow P_2, (P_2) \longrightarrow P_3, \ldots$, а (P_1) остается при этом неизменным;
- реверсирование – $(P_1) \longrightarrow P_2, (P_2) \longrightarrow P_1$, что удобно для выполнения некоторых операций.

Главное преимущество использования магазинной памяти состоит в том, что при переходе к подпрограммам (ПП) или в случае прерывания нет необходимости в специальных действиях по сохранению содержимого арифметических регистров в памяти. Новая программа может немедленно начать работу. При введении в стековую память новой информации данные, соответствующие предыдущей программе, автоматически продвигаются вниз. Они возвращаются обратно, когда новая программа закончит вычисления.

Наряду с указанными преимуществами стековой памяти также отмечают:

- уменьшение количества обращений к памяти;
- упрощение способа обращения к ПП и обработки прерываний.

Недостатки стековой организации памяти:

- большое число регистров с быстрым доступом;
- необходимость в дополнительном оборудовании, чтобы следить за переполнением стековой памяти, ибо число регистров памяти конечно;
- приспособленность главным образом для решения научных задач и в меньшей степени для систем обработки данных или управления технологическими процессами.

18 Нейрокомпьютеры. Принципы построения и функционирования. Задачи

Что касается нейрокомпьютеров, то это не только не формализуемые или плохо формализуемые задачи, в алгоритм решения которых включается процесс обучения на реальном экспериментальном материале (как правило, задачи распознавания образов), но и задачи с естественным параллелизмом (например, обработка изображений). Нейрокомпьютерные архитектуры в будущем будут завоевывать все большее место среди других архитектур. Уже сегодня существенно расширяется интерес к общематематическим задачам, решаемым в нейросетевом компьютерном базисе. Среди таких задач:

- линейные и нелинейные алгебраические уравнения;
- системы нелинейных дифференциальных уравнений;
- уравнения в частных производных;
- линейное и нелинейное программирование.

На сегодняшний день в основном сформировались три раздела нейроматематики: общий, прикладной и специальный. Нейроматематика – это раздел вычислительной математики, связанный с решением задач с помощью алгоритмов, представимых в нейросетевом логическом базисе. Прикладная нейроматематика включает задачи, в принципе не решаемые известными типами вычислительных устройств. Примерами таких задач являются:

- контроль кредитных карточек (диагностика фактической принадлежности карточки владельцу с настройкой нейронной сети в пространстве признаков покупаемых товаров);
- система скрытого обнаружения веществ с помощью устройств на базе типовых нейронов и нейрокомпьютера на заказных цифровых нейрочипах, что необычайно важно в аэропортах при обнаружении наркотиков, ядерных материалов и т. д.
- задачи обработки изображений;
- задачи обработки сигналов.

Специальная нейроматематика включает задачи повышения качества и увеличения динамических свойств изображения в системах виртуальной реальности, прямые и обратные задачи в системах защиты информации и т. д.

Логической основой нейрокомпьютеров является теория нейронных сетей (НС). Нейронная сеть – это сеть с конечным числом слов из однотипных элементов аналогов нейронов с различными типами связей между слоями нейронов. Основные преимущества НС как логического базиса алгоритмов решения сложных задач являются:

- инвариантность методов синтеза НС по отношению к размерности пространства признаков и размеров НС;
- отказоустойчивость в смысле монотонного, а не катастрофического изменения качества решения задачи в зависимости от числа вышедших из строя элементов.

Определение нейрокомпьютера наиболее реально давать, рассматривая его с точки зрения конкретной области применения. С точки зрения вычислительной техники нейрокомпьютер – это вычислительная система с архитектурой MSIMD, в которой реализованы следующие принципиальные решения:

- упрощен до уровня нейрона процессорный элемент однородной структуры;
- резко усложнены связи между элементами;

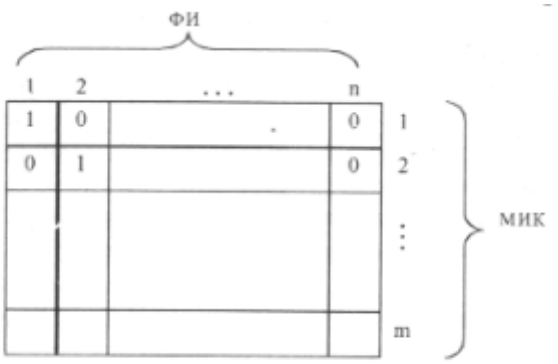
- программирование вычислительной структуры перенесено на изменение весовых связей между процессорными элементами.

Наиболее общее определение нейрокомпьютера может быть следующим. Нейрокомпьютер – это вычислительная система с архитектурой программного и аппаратного обеспечения, адекватной выполнению алгоритмов, предъявленных в нейросетевом логическом базисе.

19 Процессоры с микропрограммным управлением

Горизонтальное микропрограммирование

При горизонтальном – каждому разряду микрокоманды (МИК) соответствует определенная МИО, выполняемая независимо от содержания других разрядов. Микропрограмма может быть представлена в виде матрицы $n \times m$, где n – число функциональных импульсов (ФИ), m – количество МИК, т. е. строка соответствует одной МИК, а столбец – одной микрооперации (МИО).



Примерные значения разрядов МИК приведены на рис. 2.5.

1	2	3	4	5	6	7	8	9	...

1 – гашение сумматора; 2 – гашение указателя переполнения; 3 – обратный код сумматора; 4 – гашение регистра множителя частного; 5 – инвертирование знака; 6 – сдвиг содержимого сумматора влево; 7 – сдвиг содержимого сумматора вправо; 8 – увеличение содержимого сумматора на 1; 9 – чтение из ЗУ в сумматор и т.д.

Наличие 1 в пересечении какой-либо строки и столбца означает посылку ФИ в данную МИК, а наличие 0 – его отсутствие. Размещение 1 в нескольких разрядах МИК означает выполнение нескольких МИО одновременно. Конечно, возбуждаемые МИО должны быть совместимы.

Для расширения возможностей МИК иногда используют многотактный принцип исполнения МИК. При этом каждому разряду присваивается номер такта, в котором выполняется соответствующая ему МИО, т. с. здесь все совместимые МИО имеют один номер такта. Все остальные такты нумеруются в порядке их естественного выполнения. Однако универсальную нумерацию МИО в МИК указать затруднительно.

Достоинства горизонтального микропрограммирования:

- возможность одновременного выполнения нескольких МИО;
- простота формирования ФИ (без схем дешифрации).

Недостатки:

- большая длина МИК, так как число ФИ в современных компьютерах достигает нескольких сот;
- соответственно большой объем ЗУ для хранения МИК;

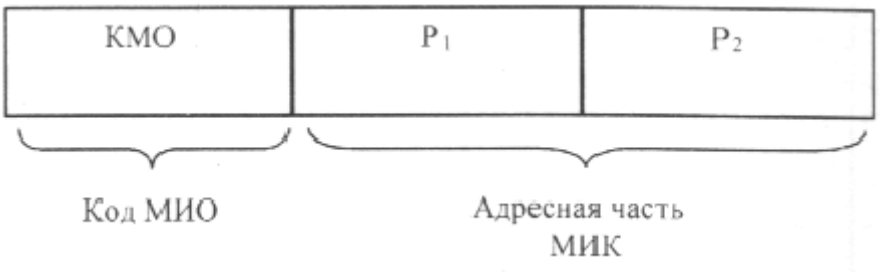
Из-за ограничений совместимости операций, а также из-за последовательного характера выполнения алгоритмов операций лишь небольшая часть разрядов МИК будет содержать 1. В основном матрица будет состоять из нулей. Неэффективное использование ЗУ привело к малому распространению горизонтального микропрограммирования.

Вертикальное программирование

При вертикальном микропрограммировании каждая МИО определяется не состоянием одного разряда, а двоичным кодом, содержащимся в определенном поле МИК. Микрокоманда несколько напоминает формат обычных команд. Отличие состоит в том, что:

- выполняется более элементарное действие МИО вместо операции;
- адресная часть (в большинстве случаев) определяет не ячейку памяти, а операционный регистр процессора.

Формат МИК при вертикальном микропрограммировании приведен ниже.



Поля P_1 и P_2 в адресной части МИК указывают двоичного номера операциных регистров, содержимое которых участвует в одной операции. Одно из полей является одновременно и адресом результата. Таким образом, реализация арифметической или логической МИО, указанной в данной МИК, может быть выражена формулой $(P_1)®(P_2) \longrightarrow P_1$ или $(P_2) \longrightarrow P_1$, где ® – символ МИО. Для МИК обращения к памяти поле P1 указывает регистр, куда принимается информация, а P2 регистр, содержимое которого является адресом обращения к ЗУ. Указанный формат МИК не единственный.

Каждая МИК выполняет следующие функции:

- указывает выполняемую МИО;
- указывает следующую МИО через задание «следующего адреса»;
- задает продолжительность МИК;
- указывает дополнительные действия – контроль и т. д.

Обычно в слове МИК имеются четыре зоны, соответствующие указанным функциям. Вообще говоря, некоторые из зон могут указываться неявно, например выбор очередной МИК может осуществляться из следующей ячейки, продолжительность МИК может быть определена одинаковой для всех МИК и т. Д

20 Повышение эффективности вычислений в компьютерах

Оказывается, что существенное влияние на повышение эффективности функционирования компьютеров оказывает топология размещения компонент компьютера.

Предположим, что все элементы компьютера соеденены между собой по топологии «шина». Если P – пропускная способность шины, N – кол-во элементов, то пропускная способность любого компьютера P/N , Отсюда следует, что свойство масштабируемости отсутствует (с увеличением количества компонентов уменьшается пропускная способность)

Топология «решето», в отличии от «шины», обладает масштабируемостью.

При обработке многопроцессорных системах в предельном случае хотелось бы, что бы с увеличением количества процессоров среднее время ожидания было постоянным, а средняя пропускная способность на любом процессоре была одной и той же. Тогда легко было бы рассчитать производительность сиситем при решении конкретных задач. Для сокращения времени ожидания применяют ряд технологий.

1. *копирование данных*

   Предполагается, что чем больш мест в общей памяти компьютера будет отведено для хранения одних и тех же данных, тем быстрее в среднем будет осуществляться доступ к ним процессора.

   Одна из возможностей реализации данной технологии – кэш-память, вторая – выделение мест для хранения с различным приоритетом. При этом надо решить кто , когда, и куда будет размещать указанные данные.

2. *упреждающая выборка*, при которой данные выбираются до того, как данные понадобятся до вычисления;

3. *многопоточная обработка*

   Позволяет быстро перекюлчать процессор от одного потока к другому.

4. *неблокирующие записи*

   Как правило процессор ждет пока операция с памятью не завершится, т.е. простаивает, поэтому предлагается не ждать ,а дальше обрабатывать данные.

5. *параллельные вычисления*

При организации вычислений в многопроц. вычислениях следует учитывать организацию ПО.

21 Вычислительные парадигмы. Методы коммуникаций

Сушествует ряд вычислительных парадигм для структуризации работы большого количества потоков или независимых процессов. Рассмотрим наиболее употребительные из них.

1. *SPMD* – одна программа, несколько потоков данных;

   В данном случае все процессы выполняют одну и ту же программу, но над разными наборами данных, то есть производят одни и те же вычисления, но каждый в своем адресном пространстве.

2. *конвейер*;

   Данные поступают в первый процесс, трансформируются и передаются второму процессу для чтения, и т.д.

3. *фазированные вычисления*;

   В этом случае выполняемая работа разделяется на фазы, в качестве которых, например, могут выступать повторения цикла. На каждой фазе одновременно работают несколько процессов, но каждый из них ожидает завершения остальных процессов фазы, после чего начинается новая фаза.

4. *разделяй и властвуй*;

   Каждый начавшийся процесс порождает новые процессы, которым он передает часть работы. Причем порожденные процессы могут в свою очередь породить ряд новых процессов с передачей им части работы.

5. *порождающая работа (replicated worker)*;

   Организуется центральная очередь, рабочие процессы получают задачи из этой очереди и выполняют их. Если задача порождает новые задачи, они добавляются в центральную очередь. Завершая выполнение текущей задачи, рабочий процесс получает из центральной очереди следующую задачу.

Процессы, выполняющиеся на различных устройствах, могут взаимодействовать с помощью либо общих переменных, либо передачи сообщений.

В первом случае взаимодействующие процессы имеют доступ к общей памяти и взаимодействуют, считывая и записывая информацию в эту память. Один процесс, например, может записывать информацию в ячейку памяти, а второй – считывать эту информацию. Отображая одну и ту же страницу в адресное пространство каждого процесса в мультипроцессоре, разделяют переменные между несколькими процессами. Затем общие переменные считываются и записываются с помощью обычных команд LOAD и STORE.

Альтернативный подход – взаимодействие через передачу сообщений посредством использования примитивов send и receive. Они реализуются как системные вызовы. Один процесс отправляет сообщение, обозначая другой процесс в качестве пункта назначения. Как только второй процесс применяет примитив receive, сообщение копируется в адресное пространство получателя. Таким способом взаимодействуют процессы в мультикомпьютере, так как они не получают доступа к памяти других процессоров с помощью команд LOAD и STORE. Эти различия полностью меняют модель программирования. Очень важный вопрос при передаче сообщений – количество получателей. Простейший случай «один отправитель – один получатель» называют двухточечной передачей сообщений. Отправление сообщения всем процессам – широковещательная передача, а в случае заданного (фиксированного) набора процессов – мультнвещание.

22 Организация системы прерываний

Определение

Прерывание программы – это свойство ВС при возникновении особых событий временно прекратить выполнение текущей программы и передать управление программе, специально предусмотренной для обработки данного события.

В системе с прерыванием каждое программно-независимое событие (источник прерывания) должно, если оно может повлиять на ход обработки, сопровождаться сигналом, говорящим о его возникновении (*запросами прерывания*). Программы, затребованные запросами прерывания, назовем *прерывающими программами*, в отличие от прерванных программ, выполнявшихся компьютером до появления запросов прерывания.

Так как функции по сохранению и восстановлению состояния прерванной программы возлагаются на саму прерывающую программу, то последняя должна состоять из трех частей: подготовительной и восстановительной, обеспечивающих переход к нужной программе, и собственно прерывающей программы. По окончании работы прерывающей программы переход может быть осуществлен либо к прерванной программе, либо к другой прерывающей программе.

Так как всевозможные запросы на прерывание вырабатываются независимо и асинхронно, то возможны также ситуации:

- приход запросов последовательный;
- одновременный приход нескольких запросов;
- приход запроса во время выполнения прерывающей программы.

Должен быть организован порядок, в котором поступившие запросы удовлетворяются. Если в ВС имеются средства для обслуживания запросов в порядке присвоенного им приоритета, то такие системы прерывания называются приоритетными. Система прерываний программ (СПП), как правило, выполняет следующие основные функции:

- организуют вход в прерывающую программу;
- существляют приоритетный выбор между запросами прерывания;
- обеспечивают возврат к прерванной программе и программное изменение приоритетов программ.

Параметры эффективности системы прерывании

Для сравнения различных СПП используются чаще всего следующие параметры их функционирования:

1. время реакции – время между появлением запроса на прерывание и началом выполнения первой команды прерывающей программы (t_p). Так как t_p зависит от приоритета программы, то для характеристики системы используют время реакции для программы с наивысшим приоритетом;
2. время обслуживания прерывания – разность между полным временем выполнения прерывающей программы ($t_{\text{пр}}$) и временем выполнения всех полезных команд ($t_{\text{п}}$);
3. удельный вес прерывающих программ $\nu = t_{\text{п}}/t_{\text{пр}}$;
4. глубина прерывания – максимальное число программ, которые могут прерывать друг друга.

Ясно, чем больше глубина прерывания, тем лучше можно учесть приоритетное обслуживание. Так. если при глубине прерывания n_0 пришла $n_0 + 1$-я программа, когда выполняется n-е прерывание, причем $n + 1$-я программа с наивысшим приоритетом, то она будет принята к исполнению только после выполнения n -й программы:

Если t_p или $t_{\text{п}}$ запроса настолько велики, что запрос окажется необслуженным к моменту прихода нового запроса от того же устройства, то возникает явление, называемое *насыщением системы прерывания*. В этом случае факт посылки предыдущего запроса от данного источника будет утрачен. Параметры системы должны быть так согласованы, чтобы насыщение системы не наступило.

Вход в прерывающую программу

Наиболее простыми являются три следующих способа определения допустимого момента прерывания.

1. Метод помеченного оператора (опорных точек).

   Суть метода состоит в следующем. В специальные разряды команд, после которых допускается прерывание, записывается определенный знак, разрешающий (например, состояние 1 специального бита команды) или запрещающий (например, состояние 0) прерывание. Тогда вдоль программы можно расставить все опорные точки. Здесь желательно так расставить точки прерывания, чтобы информация, находящаяся в регистрах процессора после выполнения данной команды, дальше не использовалась. Это уменьшает время обслуживания, но увеличивает время реакции.

2. Покомандный способ.

   Здесь прерывание допускается после выполнения любой команды. Способ прост в реализации. При этом t_p уменьшается, а $t_{\text{обс}}$ увеличивается.

3. Метод быстрого реагирования.

   Прерывание допускатеся по окончании выполнения очередного такта люой команды. В данном случае $t_p \to \min, t_{\text{обс}} \to \max$, поскольку надо запоминать и восстанавливать некоторые элементы.

23 Конвейеризация. Конфликты и механизмы их обхода

Принцип конвейерной обработки основан на разбиении вычислительного процесса на несколько подпроцессов, каждый из которых выполняется в отдельном устройстве, как это имеет место при выполнении сложных операций последовательно по подоперациям на промышленном конвейере, в связи с чем этот метод и называют конвейерным.

Конвейерная обработка

Конвейерная обработка основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями. Производительность при этом возрастает благодаря тому, что одновременно на разных ступенях конвейера выполняются различные этапы нескольких команд. Такая обработка применяется во всех современных быстродействующих процессорах. Выполнение типичной команды можно разделить на следующие этапы:

1. выборка команды IF (Instruction Fetch);
2. декодирование команды ID (Instruction Decode);
3. выполнение операции EX (Execute);
4. обращение к памяти MEM (Memory);
5. сохранение (запись) результата WB (Write Back).

Для конвейеризации потока каждая команда разбивается на указанные этапы, отведя для выполнения каждого этапа один такт синхронизации, и в каждом такте начинается выполнение новой команды. Такая организация функционирования направлена на возможно более полную загрузку всех компонент системы. Однако, во-первых, не все этапы выполнения команды идентичны по времени, во-вторых, не все команды выполняются до конца (ветвления, циклы и т. д.), а в-третьих, имеется информационная зависимость между результатами операций, что приводит к тому, что не все устройства ВС работают на полную мощность. При реализации конвейерной обработки возникают ситуации, которые мешают очередной команде начать выполнение в соответствующем такте. Такие ситуации называются *конфликтными*. Существуют три класса конфликтов:

1. Структурные, которые возникают из-за конфликтов по ресурсам, когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением;
2. Конфликты по данным, возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды;
3. Конфликты по управлению, которые возникают при конвейеризации команд переходов и других команд, которые изменяют значение счетчика команд.

Конфликты в конвейере приводят к необходимости приостановки выполнения команд (pipeline stall).

Классификация конфликтов по данным

Конфликт по данным возникает всегда, если имеет место зависимость между командами, и они расположены по отношению друг к другу так близко, что совмещение операций, происходящее при конвейеризации, может привести к изменению порядка обращения к операндам. Известны три возможных конфликта по данным в зависимости от порядка операций чтения и записи. Рассмотрим две команды i и j, при этом i предшествует j. Возможны следующие конфликты:

- RAW (чтение после записи) – команда j пытается прочитать операнд-источник данных прежде, чем команда i запишет туда данные. Таким образом, j может некорректно получить старое значение. Это наиболее общий тип конфликтов, способ их преодоления – механизм «обходов»;

- WAR (запись после чтения) – j пытается записать результат в приемник прежде, чем он считывается оттуда командой i, так что i может некорректно получить новое значение. Этот тип конфликтов, как правило, не возникает в системах с централизованным управлением потоком команд, обеспечивающих выполнение команд в порядке их поступления, так как последующая запись всегда выполняется позже, чем предшествующее считывание. Особенно часто конфликты такого рода могут возникать в системах, допускающих выполнение команд не в порядке их расположения в программном коде;

- WAW (запись после записи) – команда j пытается записать операнд прежде, чем будет записан результат команды i, то есть записи заканчиваются в неверном порядке, оставляя в приемнике значение, записанное командой i, а не j. Этот тип конфликтов присутствует только в конвейерах, которые выполняют запись со многих ступеней (или позволяют команде выполняться даже в случае, когда предыдущая приостановлена).

24 Приоритетная система прерываний. Шестиуровневая система прерываний

Аппарат приоритетов предназначен для повышения эффективности использования ресурсов. Так, например, неэффективным является одновременное выполнение двух заданий, каждое из которых требует большой загрузки устройств ввода-вывода и незначительно использует центральный процессор. Приоритет задания может назначаться исходя из:

- времени ею выполнения («короткие» задания с небольшим временем выполнения имеют более высокий приоритет по сравнению с «длинными» заданиями, требующими значительных затрат времени);

- объема используемой оперативной памяти (задания, требующие большого объема памяти, не должны иметь одинаковый приоритет);

- интенсивности и объема использования других ресурсов;

- срочности выполнения;

- взаимодействия процессов;

- последствий задержки обработки процессов и т.д.

При программном распознавании причин прерывания прерывающая программа начинает анализ с запросов на прерывание, имеющих более высокий приоритет. При этом в процессе работы программным путем можно изменить приоритет запросов. При аппаратном распознавании причин прерывания указанные функции возлагаются на специальное оборудование.

Существует два понятия приоритета в прерывании программ.

Предположим, что никаких ограничений на время поступления запросов прерывания не накладывается. При одновременном поступлении нескольких запросов для немедленного удовлетворения может быпь принят только один. Этот тип приоритета, определяющий очередность рассмотрения запросов прерывания, называют *приоритетом между запросами прерываний.*

Прерывающие программы, однако, могут иметь относительно текущей программы различную степень важности, и выполняющаяся программа не может быть прервана любым запросом. Чтобы иметь возможность прервать текущую программу принятый СПП запрос должен соответствовать программе с более высоким приоритетом, нежели выполняемая в данный момент.

Этот тип приоритета определяет старшинство программ и его обычно называют *приоритетом между прерывающими программами.*

В случае простейшей аппаратной реализации приоритета между запросами прерывания может быть использован метод последовательного поиска. Таким образом, приоритет между запросами прерывания нужен лишь для выбора одного запроса из многих, а приоритет между прерывающими программами определяет фактический порядок выполнения программ.

Маска прерывания – шаблонная последовательность знаков, управляющая сохранением или исключением отдельных частей другой последовательности знаков. Для каждой прерывающей программы может быть установлена своя маска, указывающая программы, которые способны прерывать эту программу. Каждый разряд маски соответствует отдельной программе. Маски всех программ хранятся в памяти. Иногда создаются групповые маски, при которых каждый разряд воздействует на несколько уровней одновременно.

Уровни прерываний, которые управляются одним разрядом маски прерывания, образуют *класс прерывания.* Высшей степенью этой иерархии является главный тригер прерывания, выключающий (или включающий) всю СПП полностью.

Для осуществления возврата к прерванной программе необходимо полностью восстановить ее начальное состояние. Информацию, которую следует сохранять при прерывании программы, можно разделить на основную (которая запоминается всегда) и дополнительную (необходимость запоминания которой зависит от содержания прерванной программы).

Наибольшее распространение в компьютерах получили шесть уровней прерывания:

- ввод-вывод;

  Сигнализируют системе о нормальном (или ненормальном) окончании операции ввода-вывода.

- обращение к супервизору;

  Позволяет пользователю направлять работу супервизора на реализацию нужных действий (выделить дополнительную область памяти, запустить операцию ввода-вывода и т.п.).

- программный сбой;

  Возникает в результате различного рода ошибок в программе: переполнение разрядной сетки, наруше ние защиты, появление привилегированной команды в состоянии «задача» и т.п.

- внешние прерывания;

  Происходят от внешних по отношению к компьютеру объектов оператора путем нажатия определенной кнопки, от датчика времени и т.п.

- прерывание повторного пуска;

  Позволяет оператору или некоторому процессору вызвать выполнение требуемой программы.

- прерывание от схем контроля.

  Сигнализирует о неисправности оборудования и обеспечивает ее локализацию и исправление.

25 VLIW-архитектура

VLIW — Very Long Instruction Word

Вообще, быстродействие VLIW-процессора в большей степени зависит от компилятора, нежели от аппаратуры, поскольку здесь эффект от оптимизации последовательности операций превышает результат, возникающий от повышения частоты. Архитектура VLIW представляет собой одну из реализаций концепции внутреннего параллелизма в микропроцессорах.

Их быстродействие можно повысить двумя способами: увеличив либо тактовую частоту, либо количество операций, выполняемых за один такт.

В первом случае требуется применение «быстрых» технологий и таких архитектурных решений, как глубинная конвейеризация (конвейеризация в пределах одного такта, когда в каждый момент времени задействованы все логические блоки кристалла, а не отдельные его части). Для увеличения количества выполняемых за один цикл операций необходимо на одном чипе разместить множество функциональных модулей обработки и обеспечить надежное параллельное исполнение машинных инструкций, что даст возможность включить в работу все модули одновременно. Надежность в таком контексте означает, что результаты вычислений будут правильными. Планирование порядка вычислений — довольно трудная задача, которую приходится решать при проектировании современного процессора. В суперскалярных архитектурах для распознавания зависимостей между машинными инструкциями применяется специальное довольно сложное аппаратное решение. Однако размеры такого аппаратного планировщика при увеличении количества функциональных модулей обработки возрастают в геометрической прогрессии, что, в конце концов, может занять весь кристалл процессора. На самом же деле, текущие реализации VLIW тоже далеко не всегда могут похвастаться 100% заполнением пакетов — реальная загрузка около 7 команд в такте примерно столько же, сколько и лидеры среди RISC-процессоров.

При другом подходе можно передать все планирование программному обеспечению, как это делается в конструкциях с VLIW. «Умный» компилятор должен выискать в программе все инструкции, которые являются совершенно независимыми, собрать их вместе в очень длинные строки (длинные инструкции) и затем отправить на одновременное исполнение функциональными модулями, количество которых, как минимум, не меньше, чем количество операций в такой длинной команде.

Логический слой VLIW-процессора

Процессор VLIW, имеющий схему, представленную ниже, может выполнять в предельном случае восемь операций за один такт и работать при меньшей тактовой частоте намного более эффективнее существующих суперскалярных чипов. Добавочные функциональные блоки могут повысить производительность (за счет уменьшения конфликтов при распределении ресурсов), не слишком усложняя чип. Однако такое расширение ограничивается физическими возможностями: количеством портов чтения/записи, необходимых для обеспечения одновременного доступа функциональных блоков к файлу регистров, и взаимосвязей, число которых геометрически растет при увеличении количества функциональных блоков. К тому же компилятор должен распараллелить программу до необходимого уровня, чтобы обеспечить загрузку каждому блоку — это, думается, самый главный момент, ограничивающий применимость данной архитектуры.

Аппаратная реализация VLIW-процессора очень проста: несколько небольших функциональных модулей (сложения, умножения, ветвления и т.д.). подключенных к шине процессора, и несколько регистров и блоков кэш-памяти. VLIW-архитектура представляет интерес для полупроводниковой промышленности по двум причинам. Первая — теперь на кристалле больше места может быть отведено для блоков обработки, а не. скажем, для блока предсказания переходов. Вторая причина — VLIW-процессор может быть высокоскоростным, так как предельная скорость обработки определяется только внутренними особенностями самих функциональных модулей.

Принцип действия VLIW-компилятора

Такой компилятор упаковывает группы независимых операций в очень длинные слова инструкций таким способом, чтобы обеспечить быстрый запуск и более эффективное их исполнение функциональными модулями. Компилятор сначала обнаруживает все зависимости между данными, а затем определяет, как их развязать. Чаще всего это делается путем переупорядочивания всей программы — разные ее блоки перемещаются с одного места в другое.

26 Квантовые процессоры

Квантовый компьютер — вычислительное устройство, работающее на основе квантовой механики. Квантовый компьютер принципиально отличается от классических компьютеров, работающих на основе классической механики.

Любое квантовое состояние в системе из k двухуровневых квантовых элементов может находиться в некоторой когеррентной суперпозиции из 2^k булевых элементов. Данные элементы позже были названы *кубитами*.

Квантовый компьютер, как и обычный, работает с 0 и 1, но его формальные элементы производят действия в фазовом пространстве некоторый квантовой системе, используя унитарные преобразования этого пространства. Если мы пожелаем в какой-то фиксированный момент времени попробовать выяснить, в каком состояни находится квантовая система, то этого мы не узнаем точно, лишь с какой-то вероятностью. Квантовая система в один и тот же момент времени находится во всех возможных состояниях, что называют *квантовой суперпозицией* данной системы. Квантовые вычисления состоят в выполнении последовательности унитарных операций над кубитами, однако, все эти вычисления осуществляются под воздействием обычного классического компьютера, связанного со всеми блоками квантового процессора.

Упрощённая схема вычисления на квантовом компьютере выглядит так: берётся система кубитов, на которой записывается начальное состояние. Затем состояние системы или её подсистем изменяется посредством унитарных преобразований, выполняющих те или иные логические операции. В конце измеряется значение, и это результат работы компьютера. Роль проводов классического компьютера играют кубиты, а роль логических блоков классического компьютера играют унитарные преобразования. Такая концепция квантового процессора и квантовых логических вентилей была предложена Дэвидом Дойчем.

Оказывается, что для построения любого вычисления достаточно двух базовых операций. Квантовая система даёт результат, только с некоторой вероятностью являющийся правильным. Но за счёт небольшого увеличения операций в алгоритме можно сколь угодно приблизить вероятность получения правильного результата к единице.

С помощью базовых квантовых операций можно симулировать работу обычных логических элементов, из которых сделаны обычные компьютеры. Поэтому любую задачу, которая решена сейчас, квантовый компьютер решит, и почти за такое же время. Следовательно, новая схема вычислений будет не слабее нынешней.

В квантовом случае система из n кубитов находится в состоянии, являющимся суперпозицией всех базовых состояний, поэтому изменение системы касается всех 2^n базовых состояний одновременно. Теоретически новая схема может работать намного (в экспоненциальное число раз) быстрее классической.

Возможности:

1. На квантовом компьютере можно моделировать любую квантовую систему за полиномиальное число шагов. Это позволяет предсказывать свойства молекул и кристаллов, а также проектировать микроскопические электронные устройства размеров в несколько ангстрем;

2. Разложение на простые множители и аналитические теоретико-числовые задачи существенно усложнят работу криптологов. Разлагая числа на множители, можно подбирать ключи к числам, подделывать электронные подписи, расшифровывать закодированные сообщения, что сделает ненужным передачу сообщений по существующим технологиям.

Основные проблемы:

1. Необходимо обеспечить высокую точность измерений;

2. Внешния воздействия могут разрушить квантовую систему или внести в её работу существенные искажения. Для обеспечения устойчивости работы квантового процесса необходимо его элементы изолировать друг от друга и окружающей среды, а также усовершенствовать воздействие унитарных операций на квантовую схему.

27 Топологии компьютерных сетей

Топология – это конфигурация соединения элементов ЛВС. Основные топологические решения делят, как правило, на два типа:

- *широковещательные*, в которых каждый PS (Physical Signalling) передает сигналы, которые могут восприниматься всеми остальными PS. К таким конфигурациям относятся «шина», «дерево» и «звезда с пассивным центром»;

- *последовательные*, где каждый физический подуровень передает информацию только одному из PS. К таким топологиям относятся «кольцо», «цепочка», «звезда с интеллектуальным центром», «снежинка» и «сетка».

Основной тип конфигурации локальных вычислительных сетей (ЛВС) – «шина». Коммуникации между территориально распределенными устройствами в ЛВС в чем-то похожи на коммуникации между модулями в компьютерах: высокая скорость передачи данных, частая смена структуры потока, неравномерная загрузка. Основное их отличие состоит в том, что скорость передачи данных в ЛВС может быть ниже, а длительность «взрывной интенсивности» больше. Наличие высокоскоростного общего канала – наиболее характерная особенность всех новых локальных вычислительных сетей.

Из-за неравномерного характера потока данных последние обычно передаются в форме пакетов. Так как одно устройство может получать пакеты от нескольких других устройств, то адрес отправителя – неотъемлемая часть структуры пакета.

Возможность широковещательного обращения реализуется резервированием специального адреса получателя для значения «Всем». Предполагается, что пакет с этим адресом будет обрабатываться всеми устройствами. Эффект от использования шины в ЛВС состоит в том, что внутренние шины компьютера как бы увеличиваются и охватывают целую территорию. Шины обеспечивают процессору доступ к периферийным устройствам, блокам памяти или другим процессорам, находящимся на значительном расстоянии от ЭВМ, но подключенных к шине.

Сетевой контроллер управляет использованием шины, а процессор вызовов (ПВ) управляет интерфейсом с терминалами. ПВ отвечает за установку (создание) виртуального каната через сеть к порту ЭВМ и за освобождение этого канала по окончании сеанса связи с терминалом. Во время сеанса ПВ получает данные от терминала, создает пакеты для ЛВС и направляет их сетевому контроллеру, обеспечивающему передачу пакета по шине.

Шина обычно представляет собой пассивную среду и поэтому обладает очень высокой надежностью. При использовании конфигурации «шина» возникает ряд проблем:

- каждая станция должна успеть распознать свой адрес за время, меньшее, чем время передачи данных;

- любая станция должна обеспечивать достаточную мощность сигнала, посылаемого в шину, чтобы он мог достичь наиболее удаленных станций;

- так как все станции наблюдают весь трафик, шина принципиально не защищаема, поэтому к секретным данным должны быть применены специальные способы защиты.

Конфигурация типа «дерево» образуется путем соединения нескольких шин активными повторителями или пассивными размножителями.

Конфигурация типа «звезда» – это дальнейшее развитие конфигурации «дерево с корнем» с ответвлениям к каждому подключаемому устройству.

Теперь рассмотрим конфигурации последовательного типа. Здесь к передатчикам и приемникам предъявляются более низкие требования, чем в широковещательных конфигурациях.

В конфигурациях типа «кольцо» и «цепочка» для устойчивого функционирования ЛВС требуется постоянная работа всех блоков физической среды PMA (Physical Medium Attachment).

Существуют специальные программы-«сборщики мусора», которые в случае порчи отдельных станций опознают и уничтожают невостребованные пакеты. Конфигурация «кольцо» сильно

уязвима в отношении отказов, так как выход из строя какого-либо элемента кабеля останавливает работу всей сети. Некоторым выходом является конфигурация «звездообразное кольцо», все «лучи» которой содержат по две линии. Общение между станциями осуществляется через центральный блок (обычно пассивный), который используют для локализации неисправностей.

Достоинство «звездообразного кольца» – простота управления. Отметим некоторые недостатки:

- значительно увеличивается длина кабеля;
- для каждой вновь подключенной машины необходимо прокладывать свой кабель.

Конфигурация типа «цепочка», как и «кольцо», уязвима в отношении отказов и также требует регенерации сигналов каждой станцией. Передача информации через физическую среду здесь должна осуществляться в двух направлениях.

Конфигурация типа «сетка» применяется в глобальных сетях, поскольку позволяет выбирать наиболее дешевый путь для связывания абонентов. В ЛВС это менее важное достоинство, так как передающая среда не является дорогостоящим ресурсом.

28 Архитектура программного обеспечения

Архитектура программного обеспечения – это первичная организация системы, сформированная ее компонентами, отношениями между компонентами и внешней средой системы, а также принципами, определяющими дизайн и эволюцию системы.

Для успешной реализации проекта объект проектирования должен быть прежде всего адекватно описан, т.е. должны быть построены полные и непротиворечивые модели архитектуры программного обеспечения, обусловливающей совокупность структурных элементов системы и связей между ними, поведение элементов системы в процессе их взаимодействия, а также иерархию подсистем, объединяющих структурные элементы.

В 70-80-х гг. при разработке программного обеспечения достаточно широко применялись структурные методы, базирующиеся на строгих формализованных методах описания программного обеспечения и принимаемых технических решений (в настоящее время такое же распространение получают объектно-ориентированные методы). Эти методы основаны на использовании наглядных графических моделей: для описания архитектуры программного обеспечения в различных аспектах (как статической структуры, так и динамики поведения системы) используются схемы и диаграммы. Наглядность и строгость средств структурного и объекно-ориентированного анализа позволяют разработчикам и будущим пользователям системы с самого начала неформально участвовать в ее создании, обсуждать и закреплять понимание основных технических решений. Однако широкое применение этих методов и следование их рекомендациям при разработке конкретных информационных систем сдерживалось отсутствием адекватных инструментальных средств, поскольку при неавтоматизированной (ручной) разработке все их преимущества практически сведены к нулю. Ручная разработка обычно порождала следующие проблемы:

- неадекватная спецификация требований;
- неспособность обнаруживать ошибки в проектных решениях;
- низкое качество документации, снижающее эксплуатационные характеристики;
- затяжной цикл и неудовлетворительные результаты тестирования.

Перечисленные проблемы породили потребность в программно-технологических средствах специального класса – CASE-средствах, реализующих CASE-технологию создания и сопровождения программного обеспечения информационных систем. 4

Термин CASE (Computer Aided Software Engineering) имеет весьма широкое толкование. В середине 90-х, на волне распространения клиент-серверного подхода и начала его трансформации в «многозвенный клиент-сервер», призванный обеспечить централизованное развертывание

и управление общей (для клиентских приложений) бизнес-логикой, вопросы организации архитектуры программного обеспечения стали складываться в самостоятельную и достаточно обширную дисциплину. В результате, сформировалась точка зрения на архитектуру не только в приложении к конкретной программной системе, но и развился взгляд на архитектуру, как на приложение общих (generic) принципов организации программных компонент.

В итоге, уже на сегодняшний день, на фоне такого развития понимания архитектуры, накоплен целый комплекс подходов и созданы (и продолжают создаваться и развиваться!) различные архитектурные «фреймворки», то есть систематизированные комплексы методов, пракгик и инструментов, призванные в той или иной степени формализовать имеющийся в индустрии опыт.

- принцип «разделяй и властвуй»;
- принцип иерархического упорядочения – принцип организации составных частей системы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне. Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к непредсказуемым последствиям (в том числе и к провалу всего проста). Основными из этих принципов являются:
- принцип абстрагирования – выделение существенных аспектов системы и отвлечение от несущественных;
- принцип непротиворечивости – обоснованность и согласованность элементов системы;
- принцип структурирования данных – данные должны бьпь структурированы и иерархически организованы.

Принципиальное различие между структурным и объектно-ориентированным подходом заключается в способе декомпозиции системы. Объектно-ориентированный подход использует объектную декомпозицию, при этом статическая структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами. Каждый объект системы обладает своим собственным поведением, моделирующим поведение объекта реального мира.

Концептуальной основой объектно-ориентиронанного подхода является объектная модель. Основными ее элементами являются:

- абстрагирование (abstraction);
- инкапсуляция (encapsulation);
- модульность (modularity);
- иерархия (hierarchy).

Кроме основных имеются еще три дополнительных элемента, не являющихся в отличие от основных строго обязательными:

- типизация (typing);
- параллелизм (concurrency);
- устойчивость (persistence).

29 Кодирование данных с симметричным представлением цифр

В известных позиционных системах кодирования данных (СКД) для изображения положительных и отрицательных чисел используются знаки «+» и «-» В позиционных сокращенных системах кодирования, где разряды числа наряду с положительными могут принимать и отрицательные значения, знак числа явно не указывается. Особое место среди систем такого рода занимает троичная СКД. В ней введены цифры 0, 1, для обозначения чисел 0, 1, –1. Таким образом, базисные числа расположены симметрично относительно нуля. СКД является позиционной, так как значение каждой цифры в записи числа в 3 раза больше значения той же цифры в соседней позиции.

Число А в этой системе записывается в виде $A = a_n a_{n-1} \dots a_1 a_0 a_{-1} a_{-2} \dots$, где каждое a_i может принимать значения $\{0, 1, -1\}$. Это сокращенная запись полинома:

$$A = a_n 3^n + a_{n-1} 3^{n-1} + \dots + a_1 \cdot 3 + a_0 + a_{-1} \cdot 3^{-1} + a_{-2} \cdot 3^{-2} + \dots$$

Знак числа определяется знаком старшей значащей цифры троичного изображения числа.

Сложение.

Таблица сложения

a	b		
	-1	0	+1
–1	$\bar{1}$ 1	$\bar{1}$	0
0	$\bar{1}$	0	1
+1	0	1	1 $\bar{1}$

Примеры сложения чисел:

а)

35	~	1	1	0	$\bar{1}$
+15	~	1	$\bar{1}$	$\bar{1}$	0
50	~	1	$\bar{1}$	0	$\bar{1}$	$\bar{1}$

б)

35	~	1	1	0	$\bar{1}$
-15	~	$\bar{1}$	1	1	0
20	~	1	$\bar{1}$	1	$\bar{1}$

Умножение.

Таблица умножения

a	b		
	$\bar{1}$	0	1
$\bar{1}$	1	0	$\bar{1}$
0	0	0	0
1	$\bar{1}$	0	1

Пример умножения чисел:

			1	$\bar{1}$	$\bar{1}$	0	~	15
			1	$\bar{1}$	$\bar{1}$	1	~	16
			1	$\bar{1}$	$\bar{1}$	0		
		$\bar{1}$	1	1	0			
	$\bar{1}$	1	1	0				
1	$\bar{1}$	$\bar{1}$	0					
0	1	0	0	0	$\bar{1}$	0	~	240

Деление.

Если промежуточное делимое (полученное из остатка путем сноса цифры из делимого) содержит столько же разрядов, сколько делитель, то независимо от того, больше это промежуточное делимое, чем делитель, или нет, в частное пишут «+1», если первые цифры делимого и делителя совпадают, или цифру «0», если нет. Если вслед за этим получается остаток, содержащий столько же разрядов, сколько и делитель, то повторяют указанные действия, записывая новую цифру частного в тот же разряд.

Замечание. Если есть остаток, т. е. нацело деление не осуществляется, то в частном ставим «,» и продолжаем операцию деления дальше.

Благодаря тому что основание 3 нечётно, в троичной системе возможно симметричное относительно нуля расположение цифр: -1, 0, 1, с которым связаны ценные свойства:

- Естественность представления отрицательных чисел;
- Для изменения знака представляемого числа нужно изменить ненулевые цифры на симметричные.

- При суммировании большого количества чисел значение для переноса в следующий разряд растёт с увеличением количества слагаемых не линейно, а пропорционально квадратному корню числа слагаемых.

- По затратам количества знаков на представление чисел она равна троичной несимметричной системе.

Из недостатков можно отметить появление нескольких значений для одного и того же разряда частного.

30 Кодирование данных в системах с отрицательным основанием

Нега-позиционная система счисления – это позиционная система счисления с отрицательным основанием. Особенностью таких систем является отсутствие знака перед отрицательными числами и, следовательно, отсутствие правил знаков. Всякое число любой из нега-позиционных систем, отличное от 0, с нечётным числом цифр — положительно, а с чётным числом цифр — отрицательно.

Число n в системе счисления с основанием $-r$ представляется в виде линейной комбинации степеней числа $-r$:

$$x = \sum_{k=0}^{n-1} a_k(-r)^k, 0 \leqslant a_k < r, k = \overline{0, n-1}$$

где a_k – целые числа, называемые *цифрами*, n – число разрядов.

Сложение

Сложение столбиком надо делать как в обычной системе. Например, если происходит сложение в нега-десятичной системе счисления, то это надо делать как в десятичной системе счисления, но с одним исключением: если при сложении в каком-либо разряде получается число не менее 10, то надо в этот разряд записать число единиц из полученного числа а из соседнего слева разряда вычесть единицу. Если слева нет разряда, то приписать слева 19 (для нега-десятичной, для нега-троичной 12, для нега-двоичной 11).

Вычитание

Вычитание столбиком надо делать как в обычной системе. Например, если происходит вычитание в нега-десятичной счисления, то это надо делать как в десятичной системе счисления, но с одним исключением: если при вычитании в каком-либо разряде надо занять десяток, то вы это и делаете, но из соседнего слева разряда вы не вычитаете единицу, а наоборот прибавляете её туда. Если слева нет разряда, то приписать слева 1.

Умножение

Операция умножения осуществляется посредством последовательных сложений и сдвигов.

31 Кодирование данных с помощью вычетов

Пусть $p_1, p_2, \ldots, p_n$ – целые числа, $p_i > 1, (p_i, p_j) = 1, i \neq j; M = \prod_{i=1}^{n} p_i$ и пусть x_i – наименьшие неотрицательные решения системы сравнений $A \equiv x_i \pmod{p_i}$.

Кортеж $(x_1, x_2, \ldots, x_n)$ называют *кодом числа* A в системе в коде вычетов (СКВ) при заданных основаниях $p_1, p_2, \ldots, p_n$. Компоненты x_i называют *разрядами числа* A в СКВ, записывается это $A \sim (x_1, x_2, \ldots, x_n)$. Идейными корнями данная система восходит к так называемой китайской теореме об остатках.

Среди особенностей СКВ исследователи отмечали следующие:

- каждый разряд числа несет информацию обо всем числе, откуда следует независимость разрядов друг от друга;
- отсутствие переносов между разрядами при выполнении арифметических операций упрощает их выполнение;
- малоразрядность компонент числа, в связи с чем арифметические операции превращаются в однотактные, иногда просто в выборку результата из таблицы.

Арифметические операции

Рассмотрим алгоритмы арифметических операций. Пусть необходимо выполнить операцию $A \otimes B = C, A \sim (x_1, x_2, \ldots, x_n), B \sim (y_1, y_2, \ldots, y_n), C \sim (z_1, z_2, \ldots, z_n)$.

1. если $\otimes$ – сложение, тогда $z_i = x_i + y_i \pmod{p_i}$;
2. если $\otimes$ – вычитание, тогда $z_i = x_i - y_i \pmod{p_i}$;
3. если $\otimes$ – умножение, тогда $z_i = x_i \cdot y_i \pmod{p_i}$;
4. если $\otimes$ – деление, тогда $z_i = (x_i + k \cdot p_i)/y_i$, где k – целое число от 0 до $p_i - 1$, при котором происходит целочисленное деление.

   Иногда опреацию деления A/B заменяют операцией умножения A на мультипликативную инверсию от B по модулю M.

Следует отметить, что кодирование данных в СКВ в силу вышеуказанных свойств позволяет существенно уменьшить время выполнения основных арифметических операций.

Восстановление позиционного значения числа A по его коду $(x_1, x_2, \ldots, x_n)$ в СКВ осуществляется из соотношения $A = \sum_{i=1}^{n} x_i B_i - r_A M$, где B_i – ортогональные базисы, r_A – ранг числа A.

Ортогональные базисы – целые числа, удовлетворяющие соотношению

$$B_i \equiv \begin{cases} 1, \ (\bmod p_i), \\ 0, \ (\bmod p_j) \end{cases} \quad i \neq j, i,j = \overline{1,n}$$

Ищутся базисы среди чисел вида $B_i = m_i \prod\limits_{\substack{j=1 \\ j \neq i}}^{n} p_j, m_i \in \{1, 2, \ldots, p_i - 1\}$

Перевод чисел из позиционной системы счисления в СКВ может осуществляться различными способами, например, решением системы линейных сравнений $A \equiv x_i \pmod{p_i}, i = \overline{1,n}$. Тогда $(x_1, \ldots, x_n)$ – код числа в СКВ.

Основным требованиям к модулям системы является однозначность кодирования данных, что гарантируется попарной взаимной простотой модулей.

Система в коде вычетов в основном применяется в специализированных вычислительных устройствах, т.к. операции определения знака чисел и сравнения чисел обладают высокой сложностью.

32 Кодирование числовой, текстовой, графической, звуковой информации

Информация, как правило, представляет собой *действительные числа*. Для их представления есть 2 формы:

- с фиксированной точкой;
- с плавающей точкой.

Целые числа преобразуются в двоичную форму путем последовательного деления при записи в обратном порядке.

Кроме чисел мы кодируем *текстовую информацию*. Основная операция, проводимая над отдельными символами — это сравнение симолов, для этой операции важна уникальность кода и его длинны. Для кодирования текста используются различные таблицы перекодировки. Институт стандартизации США ввел в действие американскую систему кодирования информации ASCII. Аналогичные были и в других странах (в советском пространстве использовалась KOI-7).

Кодирование графической информации в основном является её разбиением на дискретные элементы (дискретизация). Чем детальнее разбит рисунок, тем он точнее будет закодирован. Основными способами представления графики для ее хранения и обработки являются растровые и векторные изображения.

Векторные рисунки – совокупность элементарные геометрических фигуры (дуги, отрезки). Для каждой линии указываются двоичные коды типа линии, толщины и цвета. Растровые изображения – совокупность точек полученных в рез дискретизации в соответствии с матричным принципом.

Пиксель – элементарная единица изображения,цвет и яркость которой можно задавать независимо от остального изображения. Чем пиксели плотнее, тем выше качество изображения.

Кодирование звуковой информации. Для электронной обработки аналогивого сигнала необходимо его преобразовать последовательность двоичных чисел, а для этого необходимо его разбить на отдельные сигналы и оцифровать. Цифровой звук – аналоговый звуковой сигнал, представленный посредством дискретных численных значений его амплитуды. Оцифровка звука включает 2 процесса:

- Выборка сигнала по времени (дискретизация);
- Квантование по амплитуде.

Методы кодирования звука основаны на том, что теоретически любой сложный звук можно разложить на последовательность гармонических сигналов разных частот, каждый из которых представляет собой синусоиду, называемую спектром исходного сигнала. Задачей кодирования является представление звука в форме аналогового или цифрового сигнала. Процесс дискретизации по времени – это процесс получения значения сигнала, который преобразует с определённым временным шагом.

33 Вычисления с числами конечной точности.

Для того, чтобы операции с плавающей точкой выполнялись корректно и данные представлялись одинаково, обеспечивая возможность переноса данных с одного компьютера на другой, комиссия IEEE (Institute of Electrical and Electronics Engineers) разработала стандарт 754. В нём определены не только представления чисел, но и правила выполнения четырёх базовых арифметических операций.

Стандарт определяет три формата:

- с одинарной точностью (32 бита);
- с удвоенной точностью (64 бита);
- с повышенной точностью (80 бит).

Форматы с одинарной и удвоенной точноктью начинаются со знакового разряда, после чего следует порядок из 8 или 11 бит соответственно, после чего следует мантисса из 23 или 52 бит. Последний формат предназначен для уменьшения ошибок округления.

Нормализованная мантисса состоит из одного неявного бита (т.к. он всегда единичный), неявного бита для двоичной запятой, после которой следуют 23 или 52 бита.

Диапазон десятичных дробей приблизительно равен $[10^{-38}; 10^{38}]$ и $[10^{-308}; 10^{308}]$ для одинарной и удвоенной точности соответственно.

Традиционные проблемы, связанные с числами с плавающей точкой (переполнение, потеря значимости, неинициированные числа) решаются следующим образом: введены ненормализованные числа, имеющия порядок 0 и мантиссу, представленную следующими 23 или 52 битами. Неявный бит 1 слева от двоичной запятой становится 0. Основное отличие таких чисел от нормализованных – у нормализованных порядок не может быть равен нулю. Нуль имеет нулевой порядок и нулевую мантиссу.

Для обозначения переполнения введён тип *бесконечность*. Это число можно использовать в арифметических операциях: бесконечность и любое число в сумме дадут бесконечность, любое число при делении на бесконечность даёт нуль, любое число при делении на нуль стремится к бесконечности. Бесконечность имеет порядок, представленный 1 во всех битах, и нулевую мантиссу.

При делении бесконечности на бесконечность результат не определён, поэтому для такого случая существует специальный формат NaN (не число), которое также можно использовать в качестве операнда арифметической операции. Его можно использовать в случае, когда выполняется недопустимая операция, деления $0/0$, извлечения арифметического корня из отрицательного числа и т.п. Представлятся NaN порядком, содержащим 1 во всех битах, и ненулевой мантиссой.

34 Помехозащитные коды. Код Хэмминга

В зависимости от назначения и возможностей помехозащитных кодов различают следующие типы:

- самоконтрорилующиеся – автоматически обнаруживают наиболее вероятные ошибки;
- самокорректирующиеся – автоматически исправляют ошибик.

Одна из основных задач теории кодирования – это снижение вероятности появления ошибок, которые могут возникнуть из-за помех в каналах связи. Помехоустойчивое кодирование базируется на том, чтобы дать избыточную информацию за счёт удлинения самого кода.

Избыточный код, кроме m информационных разрядов, должен содержать ещё и k контрольных разрядов, причём количество контрольных разрядов характеризует абсолютную избыточность, а сумма $n = m + k$ – относительную избыточность.

Во всех помехозащитных кодах минимальное кодовое расстояние d должно быть больше единицы, при этом для обнаружении ошибки кратности t требуется, чтобы $d \geqslant t + 1$, для её исправления – $d \geqslant 2t + 1$.

Код должен быть построен таким образом, чтобы его избыточность была минимальна, но при этом часто встречающиеся ошибки были обнаружены. Также по возможности кодирование и декодирование не должно быть сложным в реализации и медленным в исполнении.

Для того, чтобы обнаружить одиночную ошибку, необходимо, чтобы код имел параметр $d \geqslant 2$, для исправления – $d \geqslant 3$. Процесс исправления состоит в том, чтобы поочерёдно изменять значения каждого разряда слова и проверять, является ли новая комбинация запрещённой. Если это так, то разряд возвращается в исходное состояние. Проводится эта процедура до тех пор, пока ошибочный разряд не будет исправлен.

Коды Хэмминга

Коды Хэмминга ориентированы на двоичную системы счисления. Они «на лету» позволяют исправлять одинарные ошибки (при $d = 3$) и обнаруживать вдобавок двукратные ошибки (при $d = 4$).

Каждый из контрольных разрядов является знаком чётности для определённой группы информационных знаков слова. При декодирования происходит k групповых проверок на чётность

и в результате каждой проверки в соответствующий разряд регистра записывается 0/1 в зависимости от результата проверки.

Группы для проверки образуются таким образом, чтобы в регистре ошибки после проверки получалось двоичное чилсло с k разрядами, указывающее на номер позиции ошибочного разряда.

Т.к. ошибка возможна в одной из n позиций, то число разрядов регистра ошибок должно удовлетворять неравенству $k \geqslant \log_2(n+1)$

35 Алгоритм деления в системе с отрицательным основанием

Пусть требуется разделить число a на число b и получить частное c, т.е. $\frac{a}{b} = c$.

Пусть b_k – число b, сдвинутое на k разрядов влево; $a_{i+1} = a_i - b_k$, причём на каждом шаге алгоритма индекс у a_i инкрементируется на 1; $N(a)$ – номер разряда старшей значащей цифры.

Пусть разрядность частного c равна $k+1$, а веса его разрядов будут иметь вид

$$(-2)^k, (-2)^{k-1}, \ldots, (-2)^0$$

Алгоритм деления

1. Сдвигаем b на k разрядов влево до тех пор, пока старшая значащая цифра у b на станет под старшей значащей цифрой a, т.е.

$$b := b_k, a = a_0$$

2. $a_1 = a_0 - b_k = a_0 + b_k + b_k(-2)$;
3. Если $N(a_{i+1}) = N(b_k)$, то записываем 1 в разряд $(-2)^k$ частного и ещё раз вычитаем b_k;
4. Если $N(a_{i+1}) < N(b_k)$, то записываем 1 в разряд $(-2)^k$ частного, меняем a_k на a_{k-1};
5. Если $N(a_{i+1}) > N(b_k)$, то записываем 0 в разряд $(-2)^k$ частного, меняем a_{i+1} на a_i, меняем a_k на a_{k-1} и производим вычитание;
6. Процесс завершается, если $a_{i+1} = 0$

Результаты 0 или 1 записывается в разряд частного $(-2)^k$, где k – индекс у b. При ситуации, когда в один и тот же разряд частного записывается несколько значений, возникается дополнительная операция сложения.

36 Процесс. Графическое представение. Форматы процессов в памяти

Процесс есть тройка (Q, f, g), где

- Q – множество состояний процесса;
- f – функция действия, $f\colon Q \leftarrow Q$;
- $g \in Q$ – начальное состояние процесса.

Действия. реализуемые процессом, будем рассматривать как результат выполнения некоторой программы на реальном процессоре.

Свойства:

- процесс не является закрытой системой и может взаимодействовать с другими процессами, воспринимая или изменяя часть среды, которую он с ними разделяет;

- каждый процесс существует лишь временно. Имеется и «главный» процесс, выполняемый вручную при включении компьютера и начинающий всю цепочку процессов;

- в любой момент процесс может быть описан его состоянием. Все параметры (переменные), характеризующие текущее состояние процесса, объединяются в «вектор состояний» или «слово состояний», которые позволяют возобновить как процесс после его прерывания, так и локальную среду.

Чаще всего процесс представляется в виде некоторого графа, включающего кружки – вершины, соединенные стрелками-дугами. По традиции (терминологической) вершины означают состояния процесса, а дуги – переходы между ними, при этом корневая вершина соответствует начальному состоянию процесса. Дуги помечаются именем (номером) события, соответствующего данному переходу

Рассмотрим структуру записей, описывающих соответствуюшие процессы, в памяти компьютера. Любой процесс, управляемый хранящейся в памяти программой, имеет два важных параметра:

- адрес, по которому выбирается следующая команда;

- адрес страницы (блока) данных, который следует прибавлять к каждой ссылке на данные во время обращения.

Периферийный процесс характеризуется типом устройства и адресом данных, участвуюших в обмене. Кроме того, процесс может быть «свободным» или «занятым», пока не выполнится определенное условие.

37 Управление процессами в многопроцессорных компьютерах

Управление процессами в многопроцессорном компьютере

Рассмотрим проекты ВС, включающие три компоненты: среду, процессоры и управление. Отметим, что центральный процессор управляется последовательностью команд, вызываемых откуда-то из среды, а периферийный процессор действует в соответствии с фиксированной, встроенной последовательностью команд.

1. Предположим, что имеется достаточное количество процессоров различного типа, чтобы обслуживать любой родившийся процесс. Управление будет состоять из центрального процессора (ЦП), управляемого процедурой, расположенной в памяти. Управляющий процессор соединен линиями связи с другими процессорами, запуская их и получая всю информацию об их состоянии. Обычное состояние ЦП - ожидание какого-нибудь события. При возбуждении управления оно определяет причину возбуждения, обрабатывает ее и. если возможно, запускает этот и/или другой процесс. Закончив обслуживание какого-то устройства, ЦП, прежде чем перейти в режим ожидания, проверяет, пуста ли очередь на обслуживание.

2. Снимем наше предположение о бесконечном количестве процессоров различных типов. Теперь процессы будут выстраиваться в очередь из-за отсутствия свободного процессора. При освобождении очередного процессора управляющий процессор (УП) должен решать, кому его предоставить. Если снабдить УП механизмом прерывания всех процессоров, то УП после сигнала об освободившемся процессоре может перераспределить работу всех процессоров заново. Этот проект оптимальнее предыдущего.

3. Из-за того, что УП часто простаивает, мы можем функции УП распределять между всеми ЦП. Центральные процессоры, работая в обычном режиме, выполняют какой-то процесс и могут быть прерваны ЦП. работающим в режиме управления. Последний не может быть прерван. Работой всех ЦП в управляющем режиме руководит одна и та же управляющая процедура.

4. Предположим, что в управляющем режиме могут работать одновременно несколько ЦП. В этом случае следует принять меры защиты информации при операциях с таблицами состояния процессов. Возбуждение управляющей процедуры (УП) может происходить одним из следующих способов:

   - если процесс, выполняющийся в ЦП, выдает сигнал связи, то этот же ЦП переключается в управляющий режим и выполняет соответствующие действия:
   - освободившийся от управления ЦП передает себя какому-нибудь процессу;
   - прерывание от периферийного устройства переводит один из ЦП в режим управления. Просмотр всех ЦП происходит поочередно, и если все они находятся в режиме у правления, прерывание помешается в очередь.

   В целях упрощения реализации можно все прерывания адресовать одному ЦП. Однако при этом увеличивается время обработки каждого события.

Управление процессами в однопроцессорном компьютере

Рассмотрим абстрактную ВС с ОП, одним ЦП, устройством ввода и устройством печати.

Пусть в исходном состоянии все периферийные устройства (ПУ) находятся в состоянии покоя, а ЦП выполняет процесс P_i, управляемый последовательностью команд.

Если в момент t_i этот процесс выдает код, возбуждающий процесс в ПУ, то ЦП переходит в управляющий режим и посылает сигнал по линии связи A_1, A_2 или A_3. Затем ЦП возвращается к выполнению своего процесса.

По окончании работы одно из ПУ посылает сигнал прерывания по линии B1, устанавливая тем самым i-й разряд регистра прерываний в соответствующее состояние. ЦП переходит в управляющий режим, анализирует состояние регистра прерываний для опознания ПУ, вызвавшего прерывания, и выполняет соответствующие действия. До перехода в режим управления ЦП запоминает состояние прерываемого процесса. Если во время обработки прерывания приходит новое прерывание, обработка предыдущего прерывания будет доведена до конца.

38 Информационные модели: мультипроцессоры и мультикомпьютеры

Предложены и реализованы две информационные модели взаимодействия: мультипроцессоры и мультикомпьютеры.

Мультипроцессоры

Если все процессоры разделяют общую физическую память, то такая система называется мультипроцессором. Мультипроцессорная модель распространяется и на ПО. Все процессы, вместе работающие на мультипроцессоре, могут разделять одно вирутуальное адресное пространство, отображаемое в общую память. Любой процесс может считывать слово из памяти или записывать слово в память с помощьюю команд LOAD и STORE. Два процесса могут обмениваться информацией, если один из них будет просто записывать данные в память, а другой считывать эти данные. Благодаря такой возможности взаимодействия двух и более процессов мультипроцессоры весьма популярны.

Выделяют 3 вида мультипроцессоров в зависимости от механизма реализации памяти совместного использования:

- UMA(Uniform Memory Access) с однородным доступом к памяти;
- NUMA – с неоднородным доступом к памяти;
- COMA (Cash ..) – 0с доступом только к кеш-памяти.

UMA обеспечивает одно и то же время доступа к слову в независимости от его расположения внутри модуля и расстояния модуля от процессора. Эта архитектура с предсказуемой производительностью.

В NUMA доступ к данным существенно зависит от расстояния процессора до модуля с необходимыми данными. NUMA может быть как с кеш-памятью, так и без неё. Наличие кеш-памяти несколько сглаживает разницу во времени доступа к информации в разных модулях. Однако если объём кеш-памяти существенно меньше объёма требуемых данных, производительность мультипроцессора снижается.

В архитектуре COMA память каждого процессора используется как кеш-память. В ней всё физическое адресное пространство делится на строки, которые мигрируют по системе по мере необходимости. Такая организация повышает частоту обращения кеш-памяти, повышая тем самым производительность системы.

Мультикомпьютеры

Параллельную архитектуру, при которой каждый процессор имеет свою собственную память, доступную только этому процессору, называют муьтикомьюером (системой с распределённой памятью).

Мультикомпьютеры обычно являются системой со слабой связью. Ключевое отличие мультикомпьютера от мультипроцессора состоит в том, что каждый процессор в мультикомпьютере имеет свою собственную уникальную память, к которой этот процессор может обращаться, выполняя команды LOAD и STORE. Мультипроцессоры имеют одно физическое адресное пространство, разделяемое всеми процессорами, а мультикомпьютеры содержат отдельное физическое адресное пространство для каждого процессора. Так как процессоры в мультикомпьютере не могут взаимодействовать друг с другом путём чтения из общей памяти и записи в общую память, необходим другой механизм взаимодействия.

Здесь процессоры рассылают друг другу сообщения, используя сеть межсоединений. При отсутствии памяти совместного использования в аппаратном обеспечении располагается определённая структура ПО. В мультикомпьютере невозможно иметь одно виртуальное адресное пространство. В мультикомпьютере для взаимодействия между процессорами часто используются примитивы SEND и RECEIVE. Поэтому ПО мультикомпьютера имеет более сложную структуру, чем программное обеспечение мультипроцессора.

При этом основной проблемой становится правильное разделение данных и разумное их размещение. В мультипроцессоре размещение частей не влияет на правильность выполнения задачи, хотя может повлиять на производительность. Мультикомьютеры проще строить, но мультипроцессоры проще программировать. Поэтому стали предприниматься попытки создания гибридных систем, которые относительно легко конструировать и легко программировать.

Это привело к различной реализации совместной памяти. Практически все исследования в области архитектур с параллельной обработкой направлены на создание гибридных форм, которое сочетает в себе преимущество обеих архитектур. Важно получить такую систему, которая была бы расширяема.

Первый подход к решению такой задачи основан на том, что современные компьютерные системы не монолитны, а состоят из ряда уровней. Это даёт возможность реализовать общую память на любом из нескольких уровней.

Второй подход: использовать аппаратное обеспечение мультикомпьютера и операционную систему, которая моделирует разделённую память, обеспечивая единое виртуальное простарнство, разбитое на страницы. При таком подходе (DSM) каждая страница расположена в одном из блоков памяти. Каждая машина содержит свою собственную виртуальную память и собственые таблицы страниц.

Третий подход: реализовать общую разделяемую память на уровне ПО. При таком подходе абстракцию разделяемой памяти создаёт язык программирования и эта абстракция реализовывается компилятором.

39 Метрика аппаратного и программного обеспечения

Основными параметрами аппаратного обеспечения метрики является параметры функционирования системы межсоединений – время ожидания и пропускная способность.

Полное время ожидания состоит из времени на отправку пакета и времени получения ответа. При посылке пакета в память измеряют время чтения и записи в память, при пересылке между процессорами измеряют время процессорной связи для пакетов минимального размера.

При коммутации каналов время ожидания состоит из времени установки и времени передачи. Для установки схемы передачи рекомендуется отправлять пробный пакет для резервирования требуемых ресурсов и передачи подтверждения (за время T_u), после чего можно отправлять готовый пакет с достаточной высокой скоростью. При пропускной способности b бит/с и объёме пакета p бит время ожидания составит $T_u + p/b$.

При пакетной коммутации не нужно посылать пробный пакет в пункт назначения, однако возникает некоторое время T_k на коммутацию пакета. Также возникает время задержки T_d, которое затрачивается на прохождение пакета через коммутатор. При наличии n коммутаторов общее время ожидания будет равно

$$T_k + n(p/b + T_d) + p/b$$

Если используется коммутация без буферизации пакетов, то время передачи пакета при всех благоприятных условиях будет приблизительно равна T_k+p/b, т.к. отсутствуют посылка пробных пакетов для установки схемы и задержки промежуточного хранения.

Очень важное значение имеет возможность быстрой передачи больших объёмов информации, что отражает суммарная пропускная способность. Это значение определяется суммированием всех пропускных способностей каналов связи. Чтобы не возникало задержек, необходимо, что пропускная способность отдельного процессора и суммарная пропускная способность каналов были согласованы.

В отношении ПО можно сказать, что в основном, всех обычных пользователей параллельных компьютеров интересует вопрос, какое ускорение они получат при запуске их программ на СПД по сравнению с однопроцессорными компьютерами. Считается, что нельзя на СПД с n процессорами получить n-кратное увеличение быстродействия.

Для достижения высокой производительности можно увеличивать количество процессоров даже при их неэффективном использовании. Система, которая увеличивает свою производительность при увеличении процессоров, называется расширяемой.

40 Алгоритмы выбора маршрутов для доставки сообщений

В любой сети межсоединений с размерностью один и выше можно выбирать, по какому пути передавать пакеты от одного узла к другому. Часто существует множество возможных маршрутов. Правило, определяющее, какую последовательность узлов должен пройти пакет при движении от исходного пункта к пункту назначения, называется *алгоритмом выбора маршрута*.

Хорошие алгоритмы выбора маршрута необходимы, поскольку часто свободными оказываются несколько путей. Хороший алгоритм поможет равномерно распределить нагрузку по каналам связи, чтобы полностью использовать имеющуюся в наличии пропускную способность. Кроме того, алгоритм выбора маршрута помогает избегать взаимоблокировки в сети межсоединений. Взаимоблокировка возникает в том случае, если при одновременной передаче нескольких пакетов ресурсы затребованы таким образом, что ни один из пакетов не может продвигаться дальше и все они блокируются навечно.

Алгоритмы выбора маршрута можно разделить на две категории: маршрутизация от источника и распределенная маршрутизация.

При *маршрутизации от источника* источник определяет весь путь по сети заранее. Этот путь выражается списком из номеров портов, которые нужно будет использовать в каждом коммутаторе по пути к пункту назначения. Если путь проходит через n коммутаторов, то первые

n байтов в каждом пакете будут содержать к номеров выходных портов, 1 байт на каждый порт. Когда пакет доходит до коммутатора, первый байт отсекается и используется для определения выходного порта. Оставшаяся часть пакета затем направляется в соответствующий порт. После каждого транзитного участка пакет становится на 1 байт короче, показывая новый номер порта, который нужно выбрать в следующий раз.

При *распределенной маршрутизации* каждый коммутатор сам решает, в какой порт отправить каждый приходящий пакет. Если выбор одинаков для каждого пакета, направленного к одному и тому же конечному пункту, то маршрутизация является *статической*. Если коммутатор при выборе принимает во внимание текущий трафик, то маршрутизация является *адаптивной*.

Популярным алгоритмом маршрутизации, который применяется для прямоугольных решеток с любым числом измерений и в котором никогда не возникает тупиковых ситуаций, является *пространственная маршрутизация*. В соответствии с этим алгоритмом пакет сначала перемещается вдоль оси x до нужной координаты, а затем вдоль оси y до нужной координаты и т. д. (в зависимости от количества измерений). Например, чтобы перейти из (3, 7, 5) в (6, 9, 8), пакет сначала должен переместиться из точки = 3 в точку = 6 через (4, 7, 5), (5, 7, 5) и (6, 7, 5). Затем он должен переместиться по оси через (6, 8, 5) и (6, 9, 5). Наконец, он должен переместиться по оси z в (6, 9, 6), (6, 9, 7) и (6, 9, 8). Такой алгоритм предотвращает тупиковые ситуации.

41 Методы синхронизации процессов

Процессы, выполняемые в мультипрограммном режиме, можно рассматривать как набор последовательных слабосвязанных процессов, которые действуют почти независимо друг от друга, лишь изредка используя общие ресурсы. Взаимосвязь между такими процессами устанавливается с помощью различных сообщений и так называемого механизма синхронизации, который позволяет согласовывать и координировать работу процессов.

Хотя каждый процесс, выполняемый в мультипрограммном режиме, имеет доступ к общим ресурсам, существует некоторая область, которую в фиксированный момент времени может использовать лишь один процесс. Нарушение этого условия приведет к неизвестному порядку обработки процессов. Назовем такую область критической. При использовании критической области возникают различные проблемы, среди которых можно выделить проблемы состязания (гонок) и тупиков (клинчей).

Условие состязания возникает, когда процессы настолько связаны между собой, что порядок их выполнения влияет на результат операции. Условие тупиков появляется, если два взаимосвязанных процесса блокируют друг друга при обращении к критической области.

В системах, допускающих перераспределение любых ресурсов в произвольной последовательности, но имеющей и не освобожденные ресурсы, время от времени должны возникать тупиковые ситуации.

Приёмы синхронизации процессов

Простейший прием - стандартные операции типа WAIT (ЖДАТЬ) и SIGNAL, (ОПОВЕСТИТЬ). Операция WAIT позволяет временно заблокировать процесс, а SIGNAL информирует систему о необходимости разблокирования процесса, задержанного из-за невыполнения условия. Следует отметить такие приемы, как БЛОКИРОВКА ПАМЯТИ (для реализации взаимного исключения одному процессу разрешается выполнить операцию над памятью, а другому ждать, пока первый не завершит работу); ПРОВЕРКА и УСТАНОВКА (аппаратная операция, к которой обращаются с двумя параметрами: ЛОКАЛЬНЫЙ И ОБЩИЙ). Наиболее эффективным и простым средством синхронизации процессов, исключающим состояние «активного» ожидания, является семафор.

Среди базовых элементов синхронизации можно назвать семафоры, блокировки, мьютексы и критические секции. Все они работают по схожему принципу: какой-то ресурс получает доступ к некоторому ресурсу, остальные же процессы этот доступ не получают. Если возникает ситуа-

ция, когда один процесс запрашивает ресурс, который уже занят другим процессом, то первый процесс получит доступ к ресурсу только после освобождения его вторым процессом.

Также используется блокирование всех процессов до тех пор, пока не выполнится какая-то фаза работы. Одним из примитивов такого рода является барьер. При встрече с барьером процесс блокируется до тех пор, пока все процессы не встретятся с барьером. Как только последний процесс достигает барьера, процессы освобождаются и продолжают работу.

42 Уровни параллелизма. Направление исследования в области параллельных вычислений

Естественный параллелизм

Говорят, что задача обладает естественным параллелизмом, если она сводится к операциям над многомерными векторами или над матрицами либо над другими аналогичными объектами.

Параллелизм множества объектов

Параллелизм множества объектов представляет собой частный случай естественного параллелизма. Задача состоит в обработке информации о различных, но однотипных объектах по одной и той же или почти по одной и той же программе. Здесь сравнительно малый вес занимают так называемые интегральные операции.

Параллелизм независимых ветвей

Присутствует, если существуют подзадачи, которые при наличии в ВС соответствующих средств могут выполняться параллельно (одновременно одна с другой).

Ветвь программы У не зависит от ветви Х, если:

- между ними нет функциональных связей, т.е. ни одна из входных переменных ветви У не является выходной переменной ветви Х либо какой-нибудь ветви, зависящей от Х;
- между ними нет связи по рабочим полям памяти;
- они должны выполняться по разным программам;
- независимы по управлению, то есть условие выполнения ветви У не должно зависеть от признаков, вырабатываемых при выполнении ветви Х или ветви, от нее зависящей.

Параллелизм смежных операций

Суть его состоит в следующем. Если подготовка исходных данных для выполнения i-ой операции заканчивается при выполнении $(i-k)$-ой операции, где $k = 2, 3, \ldots$, то одновременно с выполнением i-ой операцией можно выполнять операции $i-k+1, i-k+2, \ldots, i-1$

В последние годы интерес к параллельным вычислениям концентрируется на осмыслении общих закономерностей разработки параллельных алгоритмов. Работы в указанной области условно можно разбить на три основных направления.

1. Разработка моделей параллельных вычислений и установление соотношений между ними, определение класса задач, допускающих эффективное распараллеливание (класс NC), и класса задач, наиболее трудных для распараллеливания (P-полных задач).

2. Разработка общих методов построения параллельных алгоритмов и распараллеливание последовательных алгоритмов. Исследование сложности параллельных алгоритмов в различных областях применений.

3. Создание методов и средств отображения параллельных алгоритмов на реальные параллельные архитектуры.

Интуитивное представление о параллельных вычислениях имеет много формализаций, которые реализуются в различных моделях параллельных вычислений. Среди них:

- параллельная машина с произвольным доступом к памяти;
- машины Тьюринга;
- машины с управлением потоком данных (data flow);
- модели, базирующиеся на сетях Петри и схемах синхронизации, и т.д.

43 Языки параллельного программирования

Р-язык

Одним из первых ЯПП был матричный Р-язык. Программа, записанная на этом языке, представляет собой матрицу, элементами которой являются операторы, среди которых имеются

- операторы настройки, производящие изменение коммутаторов элементарных машин (ЭМ) либо параметров, управляющих структурой ЭМ;
- операторы обмена информацией между ЭМ;
- "пустые"операторы, пропускающие выполнение одного шага вычислений

Каждый столбец матрицы включает независимые операторы, которые можно выполнять параллельно. Процесс выполнения программы сводится к последовательному прохождению всех столбцов матрицы.

Основной принцип последовательного программирования – явное указание порядка выполнения операций – в Р-языке остается, однако здесь вводится двухкоординатная система записи, где одна координата указывает последовательность выполнения операций во времени, другая – распределение операций между ветвями вычислений. Р-язык, однако, не позволяет сохранить природную асинхронность задач и требует учитывать структуру конкретной вычислительной системы.

ЯПФ-язык

Более удобным в этом плане является язык ярусно-параллельных форм (ЯПФ). Программа на ЯПФ представляет собой мультиграф, задаваемый, например, в матричной форме. Вершинами мультиграфа являются операторы, мультиграф не имеет контуров. Дуги данного мультиграфа бывают двух типов: информационные и управляющие.

При выполнении программы на ЯПФ первоначально выделяется нулевой ярус (множество операторов, не имеющих непосредственных предшественников по информационным дугам) и инициируется выполнение операторов из этого множества. После их выполнения убираются все информационные дуги, выходящие из нулевого яруса, и в оставшемся графе таким же образом выделяется первый ярус. Процесс продолжается от яруса к ярусу. После выполнения управляющего оператора, из которого выходят управляющие дуги, происходит выбор одного из операторов, в которые заходят эти дуги. Процесс обработки ЯПФ-программы завершается, когда выполнены все операторы из яруса с наибольшим номером.

Однако представление программы в ЯПФ требует «развертки» циклических участков программы, что не позволяет использовать ее как язык практического параллельного программирования.

K-язык

В качестве языка параллельного программирования был разработан так называемый K-язык. Суть языка: задается множество элементарных операторов и множество порождающих правил построения алгоритмов, образующие некоторое исчисление. Запись вывода в этом исчислении и является K-программой. Имеются три типа основных порождающих правил:

- суперпозиция;

  Правило суперпозиции означает, что результаты выполнения операторов $b_1, b_2,....b_k$ служат аргументами для некоторого оператора B. Порядок выполнения операторов $b_1, b_2,....b_k$ безразличен, что и порождает параллелизм в их выполнении.

- дизъюнкция;

  Правило дизъюнкции задает ветвление в K- программе.

- рекурренция;

  Правило рекурренции задаёт цикличность.

K-язык относится к языкам асинхронного типа.

Язык диспозиций

Обычно для обработки некоторого алгоритма мы должны задавать последовательность операций, выполнение которых приведёт нас от исходных данных к конечному результату.

Однако существуют языки диспозиций, включающие в себя как известные алгоритмические описания, так и некоторые неалгоритмические предписания. Последние отличаются от алгоритмических тем, что кроме предписаний выполнить данное преобразование текста по некоторому правилу, используются также разрешения выполнять некоторые другие предписания в зависимости от текущего момента.

Диспозиция задаётся некоторой знаковой системой, множеством конечных операторов и графом диспозиции. Каждый оператор имеет 1 вход и несколько выходов и осуществляет преобразование исходного текста, после чего проверяет некоторые множества правил для дальнейшего выбора оператора.

Другие языки

Среди других языков параллельного программирования можно указать:

- OCCAM (специальный язык для компьютеров, представляет собой набор транспьютеров);
- Erlang (широко используется сейчас для параллельных и распределённых систем);
- различные расширения существующих языков (C, Java)

При проектировании языков параллельного программирования используют два подхода:

- изначальное проектирование языка программирования как язык параллельного типа;
- известные языки последовательного программирования расширяются дополнительными операторами, позволяющими представлять параллельные процессы.

44 Алгоритм преобразования последовательных программ в параллельные

В любом полном рассмотрении вопросов параллельного программирования должны быть представлены способы получения параллельной программы из обычной алгоритмической записи, которая несет в себе след чисто последовательной логики рассуждений, так как особые вычисления, проводимые вручную и автоматически, выполнялись последовательно. Однако оказывается, что любой реальный процесс при детальном его рассмотрении выполняется параллельно.

Как правило, основное время выполнения программ на обычных языках связано с реализацией циклов. Поэтому важной, задачей является распараллеливание циклов. Рождается проблема – преобразование последовательных программ в последовательно-параллельные. Но так как эта работа необычайно трудоемка, ее следует автоматизировать. Решение указанной проблемы позволяет

- высвободить большие трудовые ресурсы;
- использовать накопленный за долгие годы развития ЭВМ информационно-программный багаж без ручного перепрограммирования;
- осуществить переход на многопроцессорные системы с меньшими затратами труда и времени.

Распараллеливание программ можно осуществлять как на уровне отдельных задач, так и на уровне отдельных процедур, операторов, операций и микроопераций. Целесообразность преобразования на указанных уровнях должна решаться в каждом отдельном случае в зависимости от структуры ВС, типа программы и цели, которая славится при ее решении.

Рассмотрим общую схему преобразования последовательных программ в последовательно-параллельные. Организовать параллельные вычисления можно с помощью некоторой формальной процедуры, выполняемой автоматически над каждой программой, состоящей из последовательности операторов обычного ЯП. Выявление параллелизма заключается в разбиении всех операторов программ на два класса: тех, которые могут выполняться параллельно, и тех, которые должны быть завершены прежде, чем следующие последовательности операторов начнут выполняться.

Такой подход к распараллеливанию основывается на представлении программы в виде ориентированного графа G, множество вершин которого $V = \{v_1, v_2, \ldots, v_n\}$ соответствует либо отдельным операторам программы, либо совокупности этих операторов (типа подпрограмм, процедур, функций или других подобных объектов). Множество направленных дуг $U = \{u_1, u_2, .., u_m\}$ соответствует возможным переходам между операторами программы.

Построим схему алгоритма распараллеливания программ, используя некоторые сведения из теории графов.

1. Представление исходной программы в виде графа
2. Сведение циклического графа к ациклическому.
3. Наложение информационных связей, заданных между операторами программы, на связи возможных переходов.
4. Распределение вершин графа по уравнениям готовности.
5. Формирование ветвей решения.

В результате указанных действий каждый уровень разбивается на несколько подуровней и производится их согласование. Операторы исходной программы, окончательно распределенные по уровням готовности, объединяются в ветви решения, что и является окончательным результатом рассматриваемой процедуры распараллеливания.

45 Планирование в мультисистемах

Все задания (процессы), находящиеся в системе с мультипрограммированием, конкурируют из-за процессорного времени. Кроме процессов пользователя, имеются и системные процессы, для выполнения которых нужно процессорное время. Имеются, как правило, две очереди процессов:

- очередь готовых процессов – активные процессы, ожидающие кванта времени ЦП (O_r);
- очередь заблокированных процессов, ожидающих выполнения какого-нибудь события ($O_б$).

Каждый процесс p_i может находиться в одном из состояний: готовом, заблокированном или выполняемом. Если p_i принадл O_r и ему выделен квант t_{p_i} процессорного времени, то он будет выполняться, пока не произойдет одно из событий:

- истечет его квант времени;
- процесс будет заблокирован;
- процесс будет вытеснен более приоритетным процессом;
- процесс будет завершен.

Если произошло одно из указанных событий, процесс переводится либо в очередь O_r, либо в очередь $O_б$, либо оставляет систему. По окончании блокировки процессы переводятся из очереди $O_б$ в O_r. Алгоритм планирования для такой системы включает в себя способ выбора готового процесса из заданной совокупности процессов и способ вычисления величины кванта t_{p_i} для него. Каждому p_i принадлежащему O_r поставлен в соответствие приоритет – целое число, учитывающее важность процесса p_i, занимаемый им объем памяти, срочность выполнения и объем I/O, а также внешний приоритет, назначаемый пользователем.

Планирование по наивысшему приоритету

В этом методе HPF (highest priority first) процессор предоставляется тому процессу, который имеет наивысший приоритет. И если не допускается вытеснение процесса, то он выполняется до тех пор, пока не выполнится или не будет заблокирован. Если вытеснение разрешено, то поступивший процесс с более высоким приоритетом прервет выполняющийся процесс и ему будет передано управление на выполнение. Вытесненный процесс перейдет в очередь готовых процессов. Если процессор освободился, то ему из O_r назначается процесс с наивысшим приоритетом. Если очередь O_r рассортирована по приоритетам, то выбрать первый элемент просто. Если же такой сортировки нет (а при динамическом изменении приоритетов ее всегда нет), необходимо по истечении некоторого времени T проводить сортировку всей очереди O_r или его некоторой части. Здесь возможны различные модификации.

Метод круговорота (карусель)

Простой круговорот (round robin, RR) не использует никакой информации о приоритетах. Порядок обслуживания процессов следующий: первый пришедший процесс получает квант времени t_0 и встает в очередь на обслуживание, если только он себя не заблокирует. Поэтому при наличии N процессов в системе для обслуживания будем считать, что скорость работы процессора равна V/N, где V – истинное быстродействие процессора. Если $t_0 \to \infty$, то стратегия SRR (simple round robin) сводится к стратегии FIFO (first in first out). С уменьшением $t0$ улучшается обслуживание коротких процессов. Но t_0 не должно быть слишком мало: если оно становится сравнимым по порядку со временем переключения с одного процесса на другой, то задержка в выполнении процессов становится заметной.

Очереди с обратной связью

Основной алгоритм FB (feedback) очередей с обратной связью использует n очередей, каждая из которых обслуживается в порядке поступления. Новый процесс поступает в первую очередь, затем после получения кванта времени он переходит в очередь со следующим номером и так далее после очередного кванта времени. Время процессора планируется таким образом, что он обслуживает непустую очередь с наименьшим номером. В методе FB каждый, вновь поступающий на обслуживание процесс получает (в неявном виде безусловно и в соответствии со стратегией обслуживания) высокий приоритет и выполняется подряд в течение такого количества квантов времени, пока не придет следующий процесс. Но если приход нового процесса задерживается, то текущий процесс не может проработать большее количество квантов, чем предыдущий процесс.

Многоуровневое планирование

Метод многоуровневого планирования сохраняет всю информацию о процессах, выполняет перевычисления приоритета и т. д., но делает это не очень часто. В основе этого метода лежит следующий принцип – операции, которые встречаются часто, должны требовать меньше времени, чем те, которые встречаются редко. С этой целью все операции в зависимости от частоты выполнения разбиваются на уровни.

46 Классификация компьютеров

В настоящее время существует классификация многопроцессорных систем по Флинну, базирующаяся на порядке поступления потоков команд и данных на обработку их структуры образуют следующие четыре класса:

- одиночный поток команд – одиночный поток данных (SISD). Сюда относятся типичные машины последовательного действия;
- одиночный поток команд – множественный поток данных (SIMD). Это матричные и ассоциативные структуры;
- множественный поток команд – одиночный поток данных (MISD). В данный класс включаются конвейерные или магистральные структуры;
- множественный поток команд – множественный поток данных (MIMD). К этому классу принадлежат структуры, содержащие несколько процессоров, одновременно выполняющих различные фрагменты одной и той же программы.

Классификация Флинна, как видно из приведенного, базируется лишь на множестве потоков команд и данных для обработки и не является классификацией внутренней организации (архитектуры) компьютера. В отличие от Флинна систематика Шора основана на формировании компьютера из различных компонент. Выделяется шесть типов компьютеров со сквозной нумерацией.

1. М1 – одно устройство управления (УУ), обработчик (УО), память команд (ПК) и память данных (ПД). Устройство обработки может содержать множество функциональных устройств как векторного, так и скалярного типа. В связи с этим к классу М1 относят конвейерную векторную (CRAY–I) и конвейерную скалярную (СОС 7600) вычислительные системы.
2. М2 – здесь чтение разрядов данных из ПД осуществляется (в отличие от М1) из одноименных разрядов всех слов памяти. Примером могут служить системы DAP и STARAN.
3. М3 – считывание и обработка данных осуществляется в двух измерениях: по вертикальным и горизонтальным срезам памяти. Примером может служить ортогональная машина Шумана и OMEN-60.

4. M4 – включает многократно дублирующие ПД и УО, образующие процессорные элементы (ПЭ), а команды для исполнения поступают из единственного УУ. Примером этого класса может служить система PEPE без связи между отдельными ПЭ.

5. M5 – это тип M4 со связями между каждым ПЭ и его ближайшим соседом. К этому классу относится система ILLIAC IV.

6. M6 – этот тип представляет собой матрицу с функциональной логикой. Сюда можно отнести различные ассоциативные ансамбли запоминающих устройств и процессоров.