В.А. Фисун 14.12.2005

ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ

Краткий курс

1. Основные виды ЭВМ.

Электронные Вычислительные Машины (ЭВМ) разделяются на аналоговые(непрерывные) и дискретные машины, реализующие цифровые вычисления. Так как для обозначения аналоговых электронных вычислительных машин используется сокращение (АВМ), то для электронных дискретных вычислительных машин используется сокращение от ЭЦВМ - ЭВМ.

В АВМ информация может быть введена в виде физических величин и результаты могут выдаваться на самопишущие устройства или на экраны приборов. Схема такой машины строится в соответствии с математической моделью явления, описываемой изучаемый процесс. Спидометры и тахометры на щитке автомобиля - примеры тривиальных АВМ. Автопилоты самолетов создавались первоначально на принципах АВМ. Достоинство этой архитектуры в быстродействии (иногда в мгновенном срабатывании), в относительной дешевизне. К недостаткам следует отнести: низкую точность результатов, сложность программирования – настройки машины.

Другой, наиболее распространенный тип ЭВМ - цифровые вычислительные машины (ЭЦВМ) или машины дискретного действия. В машинах такого типа вся информация представляется в виде цифровых кодов и они могут называться: "компьютер", "суперкомпьютер", ”персональный компьютер”, ”PC”, ”рабочая станция”, "вычислительная система", "вычислительная среда", "универсальная ЦВМ", "ЭВМ", "вычислительная платформа". Базовые элементы вычислительной техники, классические ЭВМ - это вычислительные машины фон-Нейманновской архитектуры (Von Neumann architecture).

ВОПРОСЫ !! Но является ли ЭВМ, цифровая машиной с новыми принципами физической организации аппаратных средств, например с квантовами. А как называть пневматическую дискретную вычислительную машину?

1.1. Эволюционная классификация ЭВМ

Одна из форм классификации ЭВМ - по "поколениям" связана с эволюцией аппаратного и программного оборудования, причем основным классификационным параметром является технология производства. Классификация рассматривается на примерах из отечественной техники, что дает возможность перечислить хотя бы основных творцов отечественной информационной технологии. История отечественных исследований в данной области пока малоизвестна. Это связано с тем, что работы в данной области длительное время носили закрытый характер. В России (в СССР) начало эры вычислительной техники принято вести от 1946г., когда под руководством Сергея Алексеевича Лебедева закончен проект малой электронной счетной машины (МЭСМ - 50 оп./сек. ОЗУ на 63 команды и 31 константы) - фон

Нейманновская универсальная ЭВМ. В 1950/51 гг. она пущена в эксплуатацию. Далее, приводятся некоторые крупные отечественные достижения в области вычислительной техники.

Первое поколение ЭВМ /1946-1957гг/ использовало в качестве основного элемента электронную лампу. Быстродействие их не превышало 2-3 т. оп./сек; емкость ОЗУ - 2-4 К слов. Это ЭВМ: БЭСМ-1 (В.А. Мельников,1955г.), Минск-1 (И.С. Брук 1952/59 гг.), Урал-4 (Б. И. Рамеев), Стрела (Ю.Я. Базилевский, 1953 г.), М-20 (М.К. Сулим 1860 г.). А.Н. Мямлиным была разработана и несколько лет успешно эксплуатировалась "самая большая в мире ЭВМ этого поколения" - машина Восток. Программирование для этих машин: однозадачный, пакетный режим, машинный язык, ассемблер.

В ЭВМ **второго поколения** /1958-1964гг/ элементной базой служили транзисторы. Отечественные: Урал-14,Минск-22,БЭСМ-4,М-220,Мир-2,Наири и БЭСМ-6 (1 млн. оп./сек , 128К), Весна (В.С. Полин, В.К. Левин), М-10 (М.А. Карцев). ПС-2000,ПС-3000, УМШМ, АСВТ, Сетунь. Программирование: мультипрограммный режим, языки высокого уровня, библиотеки подпрограмм.

Элементная база ЭВМ **третьего поколения,** /1965-1971гг/ интегральные схемы - логически законченный функциональный блок, выполненный печатным монтажом. Отечественные ЭВМ этого поколения ЭВМ ЕС (Единой Системы):ЕС-1010,ЕС-1020, ЕС-1066 (2 млн. оп./сек , 8192К) и др. Программирование: мультипрограммный, диалоговый режимы, ОС, виртуальная память.

В 1996 г. в России работают 5 тысяч ЕС ЭВМ из 15 т., уставленных а СССР. НИИЦЭВТ на базе комплектующих IBM/390 разработал 23 модели производительностью от 1.5 до 167 Мфлоп (ЕС1270, ЕС1200, аналоги серверов 9672)). IBM предоставляет также лицензионные программные продукты (ОС-390). Используются в Росси для сохранения программного задела прикладных систем (проблема наследия ЕС ЭВМ).

ЭВМ **четвертого поколения** /1972-1977гг/ базируются на "больших интегральных схемах"(БИС) и "микропроцессорах". Отечественные - проект "Эльбрус", ПК. Программирование: диалоговые режимы, сетевая архитектура, экспертные системы.

ЭВМ **пятого поколения** /начиная с 1978г/ используют "сверхбольшие интегральные схемы" (СБИС). Выполненные по такой технологии процессорные элементы на одном кристалле могут быть основным компонентом различных платформ - серверов: от супер-ЭВМ (вычислительных серверов), до интеллектуальных коммутаторов в файл-серверах.

На этом поколении технологические новации приостанавливаются и в восьмидесятые годы в ряде стран появляются проекты создания новых вычислительных систем на новых архитектурных принципах. Так, в 1982 японские разработчики приступили к проекту "компьютерные системы пятого поколения", ориентируясь на принципы искусственного интеллекта, но в 1991 японское министерство труда и промышленности принимает решение о прекращении программы по компьютерам

пятого поколения; вместо этого запланировано приступить к разработке компьютеров шестого поколения на основе нейронных сетей.

В СССР под руководством А.Н. Мямлина в рамках такого проекта велась разработка вычислительной системы, состоящей из специализированных процессоров: процессоров ввода/вывода, вычислительного, символьного, архивного процессоров.

В настоящее время в России создаются мультисистемы на базе зарубежных микропроцессоров: вычислительные кластеры (НИИЦЭВТ), супер-ЭВМ МВС-1000 (В.К. Левин, А.В.Забродин). Под руководством Б.А.Бабаяна проектируется микропроцессор Мерсед-архитектуры. В.С. Бурцев разрабатывает проект суперЭМВ на принципах потоковых машин.

Эволюция отечественного программного обеспечения непосредственно связана с эволюцией архитектуры ЭВМ, первая Программирующая Программа ПП, Интерпретирующая Система- ИС создавались для М-20 (ИПМ). Для ЭВМ этого семейства были реализованы компиляторы с Алгола: ТА-1 (С.С.Лавров), ТФ-2 (М.Р.Шура-Бура), Альфа(А.П.Ершов).

Для БЭСМ-6 создан ряд операционные системы: от Д-68 до ОС ИПМ (Л.Н. Королев, В.П. Иванников, А.Н. Томилин, В.Ф.Тюрин, Н.Н. Говорун, Э.З. Любимский).

Под руководством С.С.Камынина и Э.З. Любимского был реализован проект Алмо: создание машинно-ориентированного языка и на его базе системы мобильных трансляторов.

В.Ф.Турчин предложил функциональный язык Рефал, системы программирования на базе этого языка используются при создании систем символьной обработки и в исследованиях в области мета вычислений.

2. Принципы фон-Неймана

Основные архитектурные принципы построения цифровых (дискретных) вычислительных систем (ЦВМ) были разработаны и в 1946 г. опубликованы Дж. фон Нейманом (John Louis von Neumann), Г. Голдстайном (H.H Goldstine) и А. Берксом (A.W. Burks) в отчете: "Предварительное обсуждение логического конструирования электронного вычислительного устройства". Основные принципы.

2.1. Программное управление работой ЭВМ.

Программа состоит из последовательности команд, хранимых в Оперативном Запоминающем Устройстве (ОЗУ); каждая команда задает единичный акт преобразования информации. ЭВМ по-очередно выбирает команды программы и выполняет предписанные в них дискретные вычисления. В любой момент времени работы ЭВМ выполняется только одна команда программы.

Так алгоритм вычисления площади трапеции с основаниями А и В, высотой Н {S=0.5*(A+B)*H} можно представить в виде последовательности (шагов) элементарных вычислений - команд ЭВМ (трех-адресных):

Команды	Комментарий
+, A, B,P1;	P1=A+B
*, P1,H,P2;	P2=P1*H

/,P2,''0.5'',S; S=R2/0.5.

2.2. Принцип условного перехода.

Этот принцип дает возможность перехода в процессе вычислений на тот или иной участок программы в зависимости от промежуточных, получаемых в ходе вычислений результатов. Команда условного перехода могут нарушить последовательный порядок выборки команд программы и указать команду для последующего выполнения – L в случае выполнения условий заданного соотношения. (Команды безусловного перехода нарушает порядок выбора команд всегда).

Так, определение максимального числа может быть выполнено программой:

```
   MAX = B
   IF (A<B)  GOTO L
   MAX =A
L .....
```

Команды перехода позволяет реализовывать в программе циклы с автоматическим выходом из них.

2. 3. Принцип хранимой программы

Принцип заключается в том, что команды представляются в числовой форме и хранятся в том же Оперативном Запоминающем Устройстве (ОЗУ), что и исходные данные. ОЗУ – структурно состоит из пронумерованных ячеек. Над программой можно производить арифметические действия, изменяя ее динамически.

ВОПРОС? Как можно использовать для модификации программы команду %:

%,A1,A2,A3; которая,

1. передает управление команде, размещенной в ОЗУ по адресу A2,
2. пересылает содержимое слова ОЗУ A1 в A3 (A3=A1).

2.4. Использования двоичной системы счисления для представления информации в ЭВМ.

Элементарной единицей информации является бит, принимающий одно из двух значений 0 или 1. В двоичной системе счисления представляются целые и вещественные числа над которыми ЭВМ производит вычисления, команды программ.

ВОПРОСЫ? Почему числа - степень двойки предпочтительны для измерения параметров оборудования ЭВМ. Почему нумерация строк ОЗУ начинает с нуля.

3. Структура традиционных ЭВМ

Классические (Von Neumann architecture) ЭВМ имеют следующую структуру:

АЛУ + УУ <кд.....кд.....кд→ ОЗУ,

где

- ОЗУ (Оперативное Запоминающего Устройства) - память для хранения программ и данных. Таблица, каждая строка которой содержит команду или данное в двоичной системе счисления.

- АЛУ (Арифметико- Логическое Устройство, ALU – Arithmetic and Logic Unit), устройство, которое выполняет операции над данными: аргументы и результаты операции считываются и записываются из (в) ОЗУ.
- УУ (Устройства Управления), устройство, которое последовательно выбирает команды из ОЗУ, дешифрирует их и организует выполнение заданных операций в АЛУ.
- <кд.....кд.....кд→ последовательность команд и данных, причем данные как читаются из ОЗУ, так и туда же записываются.

Совокупность АЛУ и УУ принято называть процессором (ЦПУ,CPU), резервируя слово ЭВМ (ПЭ) для полного вычислительного комплекса. (По словарю А. Синклера "processor" - блок компьютера, выполняющий вычислительные действия). В современных микропроцессорах, микросхема процессора размещается на одном кристалле (чипе) , это: УУ + АЛУ + набор регистров + кэш память. В приведенной схеме не отражены устройства ввода/вывода информации, массовая память для постоянного хранения информации.

Все современные микропроцессоры имеют фон Нейманновскую архитектуру. Для ускорения вычислений которых предложено ряд параллельных архитектур вычислительных машин, для классификации которых , можно использовать нотацию М. Флинна (M.Flynn).

4. Классификация вычислительных машин по Флинну

Данная классификация иллюстрируется схемами повышения производительности классического процессора путем увеличения количества функциональных устройств. Итак, исходная схема: АЛУ + УУ <кд.....кд.....кд> ОЗУ.

Увеличив число АЛУ, получим схему:

АЛУ + УУ <кд.....кд.....кд> ОЗУ
АЛУ <д.....д.....д> ОЗУ

По такой схеме создавалась система Эллиак 4 (отечественный аналог - ПС-2000), суперскалярные микропроцессоры.

Увеличив число УУ , то получим следующую схему:

АЛУ + УУ <кд.....кд.....кд> ОЗУ
УУ <к.....к.....к> ОЗУ

Некоторые исследователи отказывают данной схеме право на существование , другие, в качестве примера данной схемы приводят конвейерные АЛУ. (Например, сложение вещественных чисел можно реализовать последовательностью команд: выровнить мантиссы, сложить мантиссы, провести нормализацию результата.)

Наконец, можно просто умножать исходную схеиу:

АЛУ + УУ <кд.....кд.....кд> ОЗУ
АЛУ + УУ <кд.....кд.....кд> ОЗУ

По такой схеме реализуются все современные суперЭВМ

Кроме функционального различия, схемы отличаются характером нагрузки на ОЗУ-плотностью потоков команд и данных. По Флинну эта особенность и является главной

чертой и она характеризует архитектуры ЭВМ по структуре используемых потоков (Stream) команд (Instruction) и данных (Data), каждый из которых может быть одиночным и множественным. Множественный поток определяется как возможность одновременной обработки сразу нескольких элементов соответствующего потока. Комбинация признаков дает четыре вида архитектур.

- ОКОД одиночный поток команд, одиночный поток данных,
- ОКМД одиночный поток команд, множественный поток данных,
- МКОД множественный поток команд, одиночный поток данных,
- МКМД множественный поток команд, множественный поток данных.

Одозначив поток команд как '-','+' , а поток данных 'a,b' , 'c,d' эту классификацию изобразить:

```
 +  ->  ОКОД   -> a+b          +  ->              -> a+b
a,b ->                        a,b ->   ОКМД    -> c+d
                              c,d ->

 +  ->            -> a+b        + ->            -> a+b
 -  ->  МКОД    -> a-b          - ->  МКМД   ->  c-d
a,b ->                         a,b ->
                               c,d ->
```

5. Общие принципы потоковой обработки

Потоковая архитектура (data-flow) вычислительных систем обеспечивает интерпретацию алгоритмов на графах, управляемых данными. Идеи потоковой обработки информации, организации вычислений, управляемых потоками данных можно рассмотреть на примере организации ввода и суммирования трех чисел. Традиционная схема вычислений может быть представлена так: ввод (а); ввод (в); ввод (с); s = a+b; s = s+c;

Если ввод данных может быть производиться асинхронно, то организовать параллельное программирования данного алгоритма не просто. Параллельный алгоритм может быть записан в виде потока данных на графе:

```
    ввод (а)          ввод (в)      ввод (с)
          а+в       а+с          в+с
(в+с)+а           (а+с)+в          (а+в)+с
```

Здесь, начальные вершины - ввод, затем каждое введенное данное размножается на три и они передаются на вершины, обеспечивающие суммирование. Теперь, любом порядке поступления данных отсутствуют задержки вычислений для получения результата. Data-flow программы записываются в терминах графов. В вершинах графа находятся команды, состоящие, например, из оператора, двух операндов (для двуместных операций), возможно, литеральных, или шаблонов для заранее неизвестных данных и ссылки, определяющей команду - наследника и позицию

аргумента в ней для результата. Для фрагмента программы, вычисляющего оператор: a=(b+1)*(b-c), команды могут иметь вид:

L1:(+(<b>) “1” L3/1)

L2:(-(<b>) (<c>) L3/2)

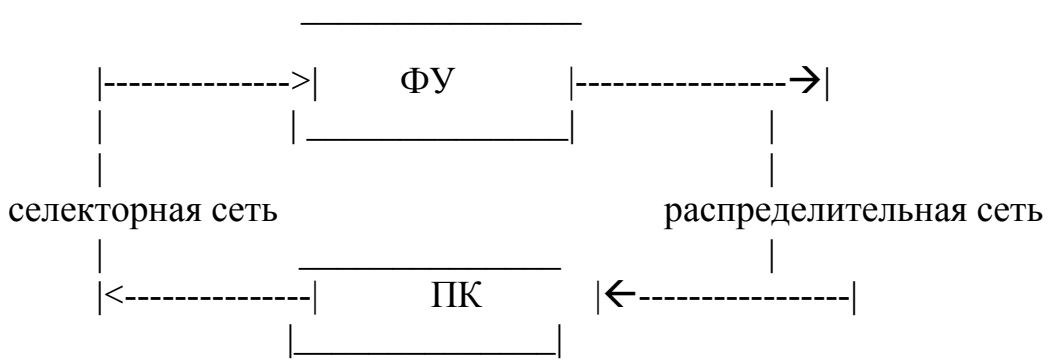
L3:(*( ) ( ) <a>)

Семантика выполнения команд следующая: операция команды Li выполняется, когда поступили все данные для их входных аргументов. Последний параметр у этих команд указывает кому и куда передавать полученные результаты (в примере это узел, команда L3, а - аргументы 1,2), в терминах графов содержит инструкцию для обмена данных.

5.1. Классическая архитектура потоковой ВС.

Основными компонентами потоковой ВС являются:

- память команд (ПК),
- селекторная (арбитражная) сеть,
- множество исполнительных (функциональных) устройств (ФУ),
- распределительная сеть.



Память команд состоит из "ячеек" активной памяти, каждая из которых может содержать одну команду вида <метка>: <операция>,<операнд1>,..,<операндК>,<адр_рез1>,..,<адр. _рез.М>, где адреса результатов являются адресами ячеек памяти. С каждой командой связан подсчитывающий элемент, непрерывно ожидающий прибытие аргументов, который пересылает команду на выполнение при наличии полного комплекта аргументов. Активных характер памяти заключается в том, что ячейка, обладающая полным набором операндов, переходит в возбужденное состояние и передает в селекторную сеть информационный пакет, содержащий необходимую числовую и связующую информацию о команде.

Селекторная сеть обеспечивает маршрут от каждой командной ячейки к выбранному, в соответствии с кодом операции, исполнительному (функциональному) устройству из множества устройств. Пакет поступает на одно из исполнительных устройств, где соответствующая операция выполняется и результат подается в распределительную сеть.

Распределительная сеть обрабатывает результирующий пакет, состоящий из результатов вычислений и адресов назначения. В зависимости от содержимого пакета, результат вычислений поступает в соответствующие ячейки памяти команд, создавая, тем самым, условия возможности их активизации.

Потоковая архитектура (data-flow), как одна из альтернатив фон-Неймановской, обладает следующими характерными чертами:

- отсутствие памяти как пассивного устройства, хранящего потребляемую информацию,
- отсутствие счетчика команд (и, следовательно, последовательной обработки команд программы, разветвлений по условию и т.д.).

Потоковые вычислительные системы позволяют использовать параллелизм вычислительных алгоритмов различных уровней, потенциально достигать производительность, недоступную традиционным вычислительным системам. Основные проблемы, препятствующие развитию потоковых машин:

1. Не решена проблема создания активной памяти большого объема, допускающей одновременную активизацию большого количества операций.
2. Создание широкополосных распределительных и селекторных сетей потоковых машин и систем управления коммуникационной сетью является сложной задачей.
3. Обработка векторных регулярных структур через механизмы потока данных менее эффективна, чем традиционные решения.
4. Языки программирования для потоковых машин существуют, в основном, в виде графических языков машинного уровня. Языки типа SISAL, ориентируемые на описания потоковых алгоритмов, достаточно сложны для программистов.

6. Ассоциативная память

Оперативную память (ОП) можно представить в виде двумерной таблицы, строки которой хранят в двоичном коде команды и данные. Обращения за содержимом строки производится заданием номера (адреса) нужной строки. При записи, кроме адреса строки указывается регистр, содержимое которого следует записать в эту строку. Запись занимает больше времени, чем чтение. Пусть в памяти из трех строк хранятся номера телефонов.

1924021
9448304
3336167

Для получения номера телефона второго абонента следует обратиться: load (2) и получить в регистре ответа R = 9448304. Такой вид памяти, при котором необходимая информация определяется номером строки памяти, называется адресной. Другой вид оперативной памяти – ассоциативной можно рассматривать также как двумерную таблицу, но у каждой строки которой есть дополнительное поле, поле ключа. Например:

Ключ	Содержимое
Иванов	1924021

Петров 9448304
Сидоров 3336167

После обращение к ассоциативной памяти с запросом : load (Петров) для данного примера получим ответ: R = 9448304. Здесь задание координаты строки памяти производится не по адресу, а по ключу. Но при состоянии ассоциативной памяти:

Ключ	Содержимое
1	1924021
2	9448304
3	3336167

можно получить номер телефона из второй строки запросом: load (2). Таким образом на ассоциативной памяти можно моделировать работу адресной. Ассоциативная память имеет очевидное преимущества перед адресной, однако, у нее есть большой недостаток - ее аппаратная реализация невозможна для памяти большого объема.

ВОПРОС !!!! Предложите схему реализации модели ассоциативной памяти при помощи адресной.

7. Адресация памяти

Адресация памяти производится с точностью до байта, длина адреса, его разрядность, определяет пространство памяти, которое может быть доступно ("видимо") в программе. Так 32 рр. виртуальный адрес охватывает пространство в 4 Гбайта. Это виртуальное пространство - математическая память программы может не совпадать с реальным, физическим пространством памяти ЭВМ.

Адреса виртуального пространства памяти - виртуальные адреса, а адреса физического пространства - физические адреса.

Механизм виртуальной памяти позволяет:

- снять ограничения, связанные с объемом памяти, при разработки алгоритмов;
- предоставлять программисту область памяти в виде логически непрерывного пространства;
- способствовать более эффективному управлению физической памятью.

Процесс преобразование виртуальных адресов в физические при выполнении программы называется трансляцией адресов, наиболее распространенный механизм для этого - страничная память. Механизмы виртуальной памяти реализуется путем разбиения памяти и виртуальной и физической на одинаковые страницы, обычно, размером 4 Кбайта. Адрес разделяется на две части в соответствии с принятой длиной страницы: номер страницы (а) и адрес внутри страницы - сдвиг, смещение (б). Трансляция адреса

Виртуальный адрес		Физический адрес	
а	б - >	с	б

Аппаратно трансляция адресов производится при помощи таблицы страниц. Каждой странице виртуальной памяти соответствует строка в таблице страниц, объем которой соответствует числу страниц виртуальной памяти. В i строке таблицы хранится: N страницы (блока) физической памяти, которая соответствует данной виртуальной, статус доступа (чтение, запись), признак записи. Полный физический адрес получается добавлением к физическому адресу, полученного из таблице страниц, смещения внутри страницы (б). Реальная структура таблицы страниц имеет более сложный вид.

8 Чередование адресов(расслоение) памяти (interleaved memory)

Расслоение памяти: память делится на M банков с автономным управлением так, что при последовательной выборки повторное обращение к одному банку произойдет через M обращений, поэтому возможно совмещение времени выборки. Для банков одинаковой емкости:B1,B2,B3,..Bm-1 адрес i трансформируется в адрес d внутри банка Bb расчетом:

i=d * m + b, где d=>0, 0<=b<=m-1

При расслоении на четыре распределение адресов в банках будет:

Адреса в банках-b	Банк 1	Банк 2	Банк 3	Банк 4	
0	0	1	2	3	Распределение
1	4	5	6	7	адресов i
2	8	9	10	11	

При конвейерном доступе при M-кратным расслоении и регулярной выборке доступ к памяти возможен в интервале 1/M цикла памяти. Но возможны конфликты по доступу, если шаг регулярной выборки коррелируется с числом банков памяти.

9. Назначение и структура кэш-памяти.

Для ускорения доступа к оперативной памяти используется буферизация данных и объектного кода в памяти, скорость которой значительно выше основной. Если бы доступ к любым типам данных был случайным, то буферизация была бы бесполезным. Эффект от буферизации можно определить среднему времени выборки: t = t2*p+ t1*(1-p), где t1 - среднее время доступа к данным основной памяти, t2 - среднее время доступа к данным из буфера (t2<t1), p - вероятность наличия данного в буфере. Очевидно, среднее время зависит от вероятности p и изменяется от среднего времени доступа к основной памяти (при p=0) до среднего времени доступа непосредственно в буфер (при p=1). Кэш (cache, cache memory) память, как правило, на порядок более быстрая, чем основная, размещается в качестве буферной, между процессором и основной памятью и служит для временного хранения (в рамках своего объема) всех данных, потребляемых или генерирумых процессором. В много-уровневых кэшах элементами связки:

"процессор - основная память" могут выступать сами кеши. Алгоритм кэширования состоит в следующем:
1. По каждому запросу процессора происходит поиск требуемого данного в кэш памяти (места для записи генерируемого данного).
2. Если данное (место) есть в кэше - кэш попадание (cache hit), то оно передается в процессор (из процессора).
3. Если нужного данного нет в кэше - кэш промах (cache miss), данное из основной памяти пересылается в кэш память, и передаются также процессору. При переполнении кэша (нет места для записи), из него удаляются (модифицированные данные сохраняются в основной памяти) часть данных, обычно, наименее востребованные.

Традиционным кэшем является "процессорный" кэш или кэш первого уровня (первичный) или кэш (L1 Cache), имеющийся на любом микропроцессоре. Это буферная память объемом от 4 Кбайт до 16 Мбайт, в которой размещаются все данные, адресуемые процессором, и из которой данные поставляются процессору. Эта память значительно быстрее основной, но меньшего объема, поэтому механизм кэширования обеспечивает обновление кэша, обычно, сохраняя в нем только наиболее часто употребляемые данные. Обмен между основной и кэш памятями производится квантами, объемами 4 - 128 байт - копируются "строки кэша" (cache line), содержащие адресуемое данное.

Обычно, программный код кешируется через особый, I кэш память, отделенной от кэша данных D-кэша. Выборка данных из кэша (hit time) прозводится, обычно, за один такт синхронизации (оценки 1 - 4 такта), потери при кеш промахе оцениваются в 8 - 32 такта синхронизации, доля промахов (miss rate) - 1% -20%. По определению, эффект кэширования основан на предположении о многократном использовании данных (Data reuse) из кэш памяти. Принято различать две формы многократного использования данных кэша:- временное использование (temporal reuse). - пространственное использование (spatial reuse). Временное использование означает, что некоторое данное, загруженные в кэш, может использоваться, по крайней мере, более двух раз. Пространственное использование кэша предполагает возможность использовать некоторый пространственный набор данных - строки кэша. Архитектура кэш памяти: полностью ассоциативная, частично ассоциативная и кэш память с прямым отображением.

10. Кэш память с прямым отображением.

Архитектура кэш-памяти с прямым отображением (direct-mapped) характеризуется наличием явной зависимости между адресами буферной и оперативной памяти, причем каждому адресу кэша соответствует адреса оперативной памяти, кратные размеру кэш памяти. Память кэша состоит их памяти данных и памяти тэгов. Пусть, длина строки кэша равна 32 байтам, размер кэша – 4 Кбайт, тогда кэш память для данных состоит из 128 строк. ОЗУ разделено на блоки, размером в 4Кбайт, каждый содержит 128 строк. В нулевой строке кэша может быть размещены 0 или 128, или .... строки ОЗУ, в первой – 1 или 129 или.. строки т и.д.

Для каждой заполненной строки данных кэша известен тэг – номер блока ОЗУ, которому принадлежит строка. Тэги хранятся в специальной памяти - памяти тэгов, размер которой – 128 строк. Длина строки памяти тэгов зависит от размера ОЗУ. Если объем ОЗУ – 4 Гбайт, тогда полный адрес - 32 бита можно представить в виде полей: 20 рр – тэг (Т), 7 рр – номер строки таблиц кэша (S), 5 рр – номер байта в строке (N). Поиск запрошенного байта (T-S-N) в кэше с прямым распределением производится так:

1.Из памяти данных и памяти тэгов кэша одновременно считываются S-ные строки.

2 Если содержимое считанной строки памяти тэгов равно Т – кэш попадание, это значит, что считанная S строка памяти данных кэша содержит запрашиваемый байт и его номер в строке есть N.

3. Если содержимое считанной строки памяти тэгов не равно Т – кэш промах, и тогда T-S строка ОЗУ переписывается в S строку памяти данных кэша, а Т записывается в S строку памяти тэгов. Затем, см по п. 1.

Кэш память Pentium III (Celeron).

Кэш память процессоров данного типа состоит из кэшей 1и 2 уровней.

Кэш первого уровня (L1) размером в 32 Кбайта разделен на кэш кода- I кэш память, и кеш данных – D кэш по 16 Кбайтов каждый. Каждый кэш (I и D) выполнен как частично ассоциативный с коэффициентом четыре (4-way). Строки кеша состоят из 32 байтов. Кэш (16 Кбайтов) поделен на четыре банка по 4 Кбайта (128 строк), каждый из которых есть кэш с прямым распределением, при этом, любой строке ОЗУ может соответствовать одна из четырех соответствующих строк банков кэша. Соответсвенно, имеются четыре памяти тегов по 128 строк. При запросе данного из процессора (T-S-N) одновременно считываются четыре S строки данных кэша и четыре S строки тэгов. Если содержимое одной из считанных строк тэга равно Т, кэш попадание и запрашиваемый байт выбирается из S строки данных того банка кэша, у которого строка тегов совпала с Т. Если содержимое строк памяти тэгов не равно Т – кэш промах. Тогда, по алгоритму LRU выбирается один из четырех банков кэша, и в него T-S строка ОЗУ переписывается в S строку памяти данных кэша, а Т записывается в S строку памяти тэгов.

Для кэша второго уровня нет разделения на I кэш память, и кеш данных – D кэш. Размер кэша может быть для различных моделей процессора составлять 128,256,512, .. Кбайт, он также частично ассоциативный с коэффициентом четыре (4-way), при этом размер банков будет соответственно равен 32,64,128 Кбайт. Строки кэша 32 байта.

11. Стратегии обновления основной памяти.

Стратегии обновления основной памяти две: метод сквозной записи и метод обратной записи.

Кэш-память с немедленной (сквозной) записью в ОЗУ (write-through).

Такой кэш при операции 'запись' данного в память всегда записывает данное в основную память и ,возможно, в кэш, если он содержит отображение соответствующей ячейки памяти.

Кэш-память с отстроченной записью в ОЗУ (write-back).

При данной схеме записи данное записывается только в кэш (на соответствующее адресу основной памяти место). Обновление содержимого основной памяти производится при выталкивании строки кеша, при выборки данного по адресу другими абонентами, которым доступна память: вывод, другие процессоры. Естественно, при первом способе записи кэш-эффект отсутствует, но реализация второго способа дороже.

12. Конвейеризация вычислительных операций

Длительность арифметической операции может быть уменьшена за счет временного перекрытия ее различных фаз, путем конвейеризации вычислительной работы. Для этого механизмы Арифметического Логического Устройства (АЛУ) выполняется по конвейерному принципу. Пусть, работа АЛУ - для сложении данных, разделена на три этапа, на три автономных блока Pi : P1 - выравнивание порядков операндов, P2 - операция сложения мантисс, P3 - нормализация результата, каждый из которых выполняется за один условный такт вычислителя. И пусть на таком АЛУ выполняются вычисления:

A1 = B1+C1
A2 = B2+C2
A3 = B3+C3
A4 = B4+C4

Тогда временная диаграмма работы АЛУ имеет вид:

Устройство	1 такт	2 такт	3 такт	4 такт	5 такт	6 такт
P1	B1+C1	B2+C2	B3+C3	B4+C4	нет работы	нет работы
P2	нет работы	B1+C1	B2+C2	B3+C3	B4+C4	нет работы
P3	нет работы	нет работы	B1+C1	B2+C2	B3+C3	B4+C4

Вычисления будут выполнены за 6 тактов работы вычислителя, причем, здесь, только два такта работы оборудование было загружено полностью.

ВОПРОС !!!! За сколько тактов будет выполнены эти вычисления, если АЛУ не конвейеризовано. Сокращает ли конвейеризация время выполнения отдельной операции ?

Оптимальную загрузку конвейерных АЛУ можно получить при работе с регулярными структурами, например, по-элементное сложение векторов. В общем случае, пусть работа операционного блока разбивается на n последовательных частей (стадий, выполняемые за одинаковое время), на которых вычислительные операции выполняются в конвейерном режиме. Тогда, если на выполнение одной операции сложения блоку требуется время T, то на обработку N операций сложения время: $Tn = (n + N) * (T / n)$. Следовательно, если $n << N$, то , то ускорение вычислений будет в n раз.

Фактором, снижающим производительность конвейеров, является конфликты по данных. Так вычисления:

Вычисления	другая форма этих вычислений
A1 = B1+C1	
A2 = A1+C2	A2 = (B1+C1)+ C2
A3 = B3+C3	A3 = B3+C3
A4 = B4+C4	A4 = B4+C4

для правильной работы будут выполняться на два такта дольше:

Устр.	1такт	2такт	3такт	4такт	5такт	6такт	7такт	8такт
P1	B1+C1	н/р	н/р	A1+C2	B3+C3	B4+C4	н/р	н/р
P2	н/р	B1+C1	н/р	н/р	A1+C2	B3+C3	B4+C4	н/р
P3	н/р	н/р	B1+C1	н/р	н/р	A1+C2	B3+C3	B4+C4

На этой диаграмме видно, что количество простаивающих тактов работы оборудования “н/р” – “конвейерных пузырей “ (pipeline bubble) стало больше.

13. Внеочередное выполнение команд

Методы динамической оптимизации: неупорядоченное выполнение “out-of-order execution” , неупорядоченная выдача “out-of-order issue” основаны на изменении порядка вычислений. Так, пример вычислений из предыдущего раздела после его преобразования к виду:

A1 = B1+C1	
A3 = B3+C3	A3 = B3+C3
A4 = B4+C4	A4 = B4+C4
A2 = A1+C2	A2 = (B1+C1)+ C2

позволит проводить вычисления без пропуска рабочих тактов для ожидания вычисления. Оптимизационные преобразования последовательности вычисления могут проводиться динамически, аппаратурой АЛУ, а также, в ряде случаев и статически проводиться системами программирования.

14. Спекулятивное выполнение команд

В конвейерных архитектурах устройство выборки команд является таким же конвейером, как и другие функциональные устройства. Так, для условного операторы: IF (A<B) GO TO L;S1;L:S2 еще до вычисления значения условного выражения A<B необходимо решать задачу о заполнении конвейера команд кодами S1 или S2 – спекулятивного выполнения программы (чтобы не было пропуска тактов конвейера из за неверно выбранной ветки, коды которой потребуется убирать из конвейера).

Тривиальное решение состоит в выборе кода, текстуально следующего за командой условного перехода. Для такого оборудования компиляторы могут формировать объектный код с размещением наиболее вероятно выполняемым фрагменте программы непосредственно за командой условного перехода. Так, для циклических конструкций, вероятность перехода на повторение цикла выше вероятности выхода из него. Некоторые системы программирования дают

возможность программисту указывать вероятность перехода по метке в условном переходе.

Аппаратный механизм учета вероятности перехода состоит из блока предсказания переходов. Этот блок, кроме (вместо) статически определенных предпочтений для ветвлений, имеет таблицу переходов, в которой хранится история переходов для каждого (в рамках объема таблицы) перехода программы. Большинства современных микропроцессоров обещают точность предсказаний переходов этим способом выше 90%. Причина повышенного внимания к этому вопросу обусловлена большими задержками, возникающими при неверном предсказании переходов, что грозит существенной потерей производительности. Используемые в микропроцессорах методы предсказания переходов, как уже было сказано, бывают статические и динамические. Как динамический, так и статический подходы имеют свои преимущества и недостатки.

Статические методы предсказания используются реже. Такие предсказания делаются компилятором еще до момента выполнения программы. Соответствующие действия компилятора можно считать оптимизацией программ. Такая оптимизация может основываться на сборе информации, получаемой при тестовом прогоне программы (Profile Based Optimisation, PBO) или на эвристических оценках. Результатом деятельности компилятора являются "советы о направлении перехода", помещаемые непосредственно в коды выполняемой программы. Эти советы использует затем аппаратура во время выполнения. В случае, когда переход происходит, или наоборот - как правило, не происходит, советы компилятора часто бывают весьма точны, что ведет к отличным результатам. Преимущество статического подхода - отсутствие необходимости интегрировать на чипе дополнительную аппаратуру предсказания переходов.

Большинство производителей современных микропроцессоров снабжают их различными средствами динамического предсказания переходов, производимого на базе анализа "предыстории". Тогда аппаратура собирает статистику переходов, которая помещается в таблицу истории переходов BHT (Branch History Table).В обоих случаях компилятор не может выработать эффективные рекомендации на этапе трансляции программы. В то же время схемы динамического предсказания переходов легко справляются с такими задачами. В этом смысле динамическое предсказание переходов "мощнее" статического. Однако у динамического предсказания есть и свои слабые места - это проблемы, возникающие из-за ограниченности ресурсов для сбора статистики .

15. Векторно - конвейерные ЭВМ.

Реализация команд организации цикла (счетчик и переход) при регулярной работе с данными - накладные расходы и препятствие опережающему просмотру на обычных, скалярных вычислителях, показывает, что такие вычисления эффективнее выполнять на специализированном векторно-конвейерном вычислителе.

Если вместо цикла:

DO L=1,N A(I) = B(I)+C(I) ENDDO
использовать запись алгоритма в виде векторной команды сложения вида:
VADD(B,C,A,N), то векторный вычислитель, выполняющий такие команды, будет вырабатовать результаты на каждом такте. В таком вычислителе имеется один (или небольшое число) конвейерный процессор, выполняющий векторные команды путем засылки элементов векторов в конвейер с интервалом, равным длительности прохождения одной стадии обработки. Скорость вычислений зависит только от длительности стадии и не зависит от задержек в процессоре в целом.

Так как конвейер для однотипных операций дешевле и быстрее чем для многофукциональных , то выгодно их делать специализированными - однофункциональным: например, только для + или только для *. Для их совместного работы используется принцип зацепления конвейеров. Так, в ЭВМ Крей-1 имеется 12 конвейеров, из них 8 могут быть зацеплены, то есть результаты вычисления конвейера могут входными аргументами для другого. Операнды (результаты) находятся в памяти верхнего уровня или на регистрах. Для операндов задается: базовый адрес вектора, число элементов, тип данных в каждом элементе, схема хранения вектора в памяти. Некоторые векторные машины могут работать с двух-трех мерными массивами.

16. Производительность конвейерных вычислителей

Время выполнения отдельной скалярной операции на конвейерном вычислителе равно: T = S + K, где K - время работы, за которое конвейер выдает очередной результат, а S - время запуска конвейера, время заполнения конвейера, которое без учета времени подготовки операндов, равно: S = K*(m-1), где m - число ступеней конвейера. Производительность конвейерного вычислителя на скалярных операциях (число результатов, выдаваемых за единицу времени) равна: R = 1/(S + K).

Время выполнения векторной операции на конвейерном вычислителе равно: T = S + K*N, где N - длина вектора. Производительность конвейерного вычислителя при векторной работе (число результатов, выдаваемых за единицу времени) равна: R = N/(S + K*N), асимптотическая производительность Rб = 1/K.

Например, при K = 10 нс, Rб = 10**8 результатов/сек, т.е. 100 мегафлопов. Графики достижения такой производительности для S = 100 нс. и S= 1000 нс. показывают, что они имеют различное расстояние до асимптоты. Для оценки этого эффекта используется величина N1/2, определяемая как длина вектора, для которой достигается половина асимптотыческой зависимости. Для приведенного выше примера N1/2 = 100 для S =1000 и N1/2 = 10 для S = 100.

17. Системы команд процессоров

Традиционные процессоры универсальных ЭВМ называются CISC (Complex (Complicated) Instruction Set Computer) - компьютерами с полной системой команд (Семейства Intel ЭВМ от серии ix86 до Pentium и Pentium Pro). Помимо арифметических и логических операций в систему команд (аппаратно реализуемые

функции) включались сложные операции, такие как извлечения корня; реализуются сложные схемы формирования исполнительных адресов (в i486 доступно до 12 способов). Расширение спектра операций облегчает программирование и отладку программ, однако увеличивает трудоемкость разработки процессоров. Время выполнения даже таких операций, как умножение и деление чисел с плавающей запятой варьюируется в зависимости от значений аргументов операций, что ограничивает возможности аппаратного планирования совмещенного выполнения команд.

CISC набор команд нельзя было разместить целиком на одном кристалле – чипе первых СБИС. Исполнение процессоров на кристалле потребовало ревизии системы команд: что привело к реализации компьютеров с сокращенным набором команд - RISC (Reduced Inctruction Set Computing) архитектуры. Традиционное число команд здесь - 128. Выполнение команд, не входящие в этот состав производится программно, для этого в архитектуре предусматривается развитый механизм реализации подпрограмм: реализация запоминания-восстановления регистров, стеки и т.д. Упрощение оборудования, конвейеризация выполнения команд позволило RISC процессорам резко повысить производительность. Недостатки данной архитектуры: отсутствие операций регистр-память вызывают необходимость подкачки данных командами обмена между регистровым файлом процессора и оперативной памятью, а ограниченный формат команд не позволяют минимизировать объем исполняемого кода. Наиболее известны следующие типовые архитектуры RISC процессоров микропроцессоров(МП): PowerPC, PA-RISC, Alpha, SPARC (Scalable Processor Architecture) - МП с изменяемой архитектурой, разработанный в 80 годы в Калифорнийском университете (Беркли) MIPS (Mikroprocessor without Interlocked Pipeline Stages) - МП без блокировки конвейера, разработанный в 80 годы в Стенфордском университете, архитектура запатентована в 1990г.

18. Сверхдлинные команды

В ЭВМ с архитектурой VLIW (Very Long Instruction Word) - (очень большие командные слова), команды могут иметь широкий формат (длину) и команда может содержать несколько содержательных инструкций, выполнение которых детально регламентируется в терминах тактов работы АУ (параллельное выполнение нескольких команд в АУ). В таких архитектурах имеется возможность программировать вычислительные алгоритмы (включая векторные) с максимальной производительностью для данной аппаратуры. В них вся работа по оптимальному программированию возлагается на системы программирования (или ручное программирование).

Однако, упрощения в архитектуре управления приводит к значительному возрастанию сложности задачи планирования выдачи команд, так программными средствами должна быть обеспечена точная синхронизация считывания и записи данных. При этом необходимо так планировать параллельное выполнение операций машины, чтобы выполнялись определенные ограничения на число одновременно считываний и

записей в наборы регистров, использование ФУ и т.д. Размер командного слова в машинах данной архитектуры - FPS (AP-120B) - 64 бита, Multilow Tract - 1024.

Определяющие свойства архитектуры VLIW:

- Одно центральное управляющее устройство, обрабатывающее за один такт одну длинную команду.
- Большое число функциональных устройств (ФУ) - АЛУ.
- Наличия в длинной команде полей, каждое из которых содержит команду управления некоторым функциональным устройством или команду обращения к памяти.
- Статически определенная длительность в тактах исполнения каждой операции. Операции могут быть конвейеризованы.
- Закрепление во время компиляции банков расслоенной памяти за ФУ для получения максимальной ширины доступа для данных, которые можно соединить в одну команду.
- Система передвижения данных между ФУ, минуя память. Маршрут передвижения полностью специфицируется во время компиляции.
- Практическая невозможность ручного программирования в силу большой сложности возникающих комбинаторных задач.

19. Производительность вычислительных систем.

Оценка производительности вычислительных систем имеет два аспекта: оценка пиковой производительности – номинального быстродействия системы и получение оценок максимальной - “реальной” производительности. Если номинальная(пиковая, предельная) оценка однозначно определяется техническими параметрами оборудования, то вторая характеристика указывает производительность системы в реальной среде применения. Для оценки производительности вычислительных систем в TOP500 используются обозначения Rpeak – пиковая, предельная производительность и Rmax – максимальная производительность при решении задачи Linpack.

19.1. Единицы измерения быстродействия ЭВМ

Наиболее абстрактной единицей измерения производительности микропроцессоров является тактовая частота ПК, частота тактового генератора (*clock rate,).* Любая операция в процессоре не может быть выполнена быстрее, чем за один такт (период) генератора. Итак, минимальное время исполнения одной логической операции (переключение транзистора) - один такт. Тактовая частота измеряется в Герцах – число тактов в секунду. 1 МГерц – миллион тактов в секунду, ГГерц – миллиард тактов в секунду. С наибольшей частотой работают элементы ПК, интегрированные на чипе – АЛУ, кэш память, дешифратор команд, регистры, а частота синхронизации пересылки между кэшем и памятью (ОЗУ) ниже. Разрядность шины адреса для большинства процессоров –32 байта и она должна работать с частотой кэш памяти. Шины данных работает на частоте работы основной памяти. При этом частоты работы

шины данных, например, равны 66, 66, 166 Мгц для микропроцессоров Pentium Pro-200, Power PC 604E-225, Alpha 21164-500, работающих на тактовых частотах 300, 225, 500 Мгц соответственно. При ширине шин 64, 64, 128 разрядов это обеспечивает пропускную способность интерфейса с основной памятью 512, 512, 2560 Мбайт/с, соответственно.

Внутренняя архитектура процессора, как и тактовая частота, также влияет на работу процессора, поэтому два CPU с одинаковой тактовой частотой не обязательно будут работать одинаково. Кроме того, некоторые микропроцессоры являются *суперскалярными*, т.е. они могут выполнять более одной команды за тактовый цикл. Итак, тактовая частота ПК, даже с уточнениями частоты шины могут быть оценкой лишь номинальной производительности ПК.

Другой обобщенной мерой производительности ПК может служить число команд, выполняемые в единицу времени. Для вычислителей фон-Нейманновской архитектуры скорость выполнения команд уже может быть параметром, который может быть использован для оценки времени выполнения программы. Этот параметр - MIPS (Million Instruction Per Second)- миллион операций (команд,инструкций

ЭВМ)/сек. Так как время выполнения различных команд может различаться, то данных параметр сопровождался разного вида уточнениями (логических команд, заданной смеси команд и т.д.), а также наиболее известной здесь мерой в 1 MIPS – это производительность вычислителя VAX 11/780. Итак, данный параметр также достаточно условен.

Так как для большинства вычислительных алгоритмов существуют оценки числа арифметических операций, необходимых для выполнения расчетов, данная мера и может служить тем показателем, который и интересует пользователей в первую очередь. Это – MFLPOPS (Million of Floating point Operation Per Second – Мегафлопс)- миллион операций на данных с плавающей запятой/сек, единица быстродействия ЭВМ в операциях с плавающей запятой, есть также единицы - GFLPOPS и TFLPOPS (терафлопс = 10**12 оп./сек.).

20. Измерение реальной производительности

Обычно, рассматриваются три подхода к оценке производительности:

- на базе аналитических модели (системами массового обслуживания);
- имитационное моделирование;
- измерения.

Первый подход обеспечивает наиболее общие и наименее точные результаты, последние, наоборот, - наименее общие и наиболее точные. Измерения проводятся контрольными (тестовыми) программами.

Бенч-марк (Benchmark) - эталон:

- стандарт, по которому могут быть сделаны измерения или сравнения;

- процедура, задача или тест, которые могут быть использованы для сравнения систем или компонентов друг с другом или со стандартом как в п.1.

Для повышения общности и представительности оценки производительности контрольные программы можно разделить на:

- программы нижнего уровня. Эти программы тестируют основные машинные операции - +,/,* , с учетом времени доступа к памяти, работу кэша, характеристики ввода/вывода.
- ядра программ. Ядра программ - короткие характерные участки программ, например, Ливерморские фортрановские ядра (24 ядра) , Эймсовские ядра НАСА, синтетический тест Ветстоун (Whetstone).
- основные подпрограммы и типовые процедуры; Примером основных подпрограмм могут быть программы Линпак (Linpack) , программы типа быстрого преобразования Фурье.

Программа Линпак - процедура решения системы линейных уравнений методом исключения Гаусса. В этой схеме вычислений точно известно число операций с плавающей точкой, которое зависит от размерности массивов – параметров. Стандартные значения размерностей 100 или 1000. Для параллельных ЭВМ имеется соответствующая версия теста.

- полные основные прикладные программы; В качестве примеров программ этого уровня приводятся Лос-Аламосские тестовые программы моделирования поведения плазмы и программы гидродинамики.

- перспективные прикладные программы.

Более корректные результаты реальной производительности вычислительных систем получаются при использовании полных прикладных программ с полным же набором входных данных. Имеются тестовые пакеты с ориентацией на оценку микропроцессоров, наиболее известные, разработаны фирмой SPEC (Systems Performance Evaluation Cooperative), SPECintXXXX SPECfpXXXX (XXXX- год выпуска теста), которые характеризуют быстродействие ПК при обработке целочисленных и вещественных чисел.

21. Закон Амдала

Закон Амдала показывает коэффициент ускорения выполнения программы на параллельных системах в зависимости от степени распараллеливания программы. Пусть: N - число процессоров системы, P - доля распараллеливаемой части программы, а S = 1-P - доля скалярных операций программы, выполняемых без совмещения по времени.(S+P = 1 , S,P >= 0). Тогда, по Амдалу, общее время выполнения программы на однопроцессорном вычислителе S+P, на мультипроцессоре: S+P/N, а ускорение при этом есть функция от P: Sp = (S+P)/(S+P/N) = 1/(S+P/N)

Из формулы закона Амдала следует,что при:

P = 0 S = 1 - ускорения вычисления нет, а

P = 1 S = 0 - ускорение вычислений в N раз

Если P = S = 0.5, то даже при бесконечном числе процессоров ускорение не может быть более чем в 2 раза.

22. Архитектура мультипроцессорных систем.

Архитектуры параллельных вычислителей в свое время разделялись на многомашинные (слабосвязанные) и многопроцессорные (сильносвязанные) комплексы. По типу (“силе”) связей между элементарными вычислителями мультипроцессорные комплексы, в настоящее время, можно классифицировать: GRID-сети, кластеры, суперЭВМ (МИМД системы по Флинну) .

GRID-сети это совокупность вычислительных систем, связанных через Интернет (метакомпьютинг). Используются для решения задач, допускающих сегментацию на независимые вычислительные процессы с большим объемом вычислений. Такими задачами являются задача исследования генома, обработку результатов физических испытаний, проект CETI.

Вычислительные кластеры (cluster) – группа ЭВМ (серверов), связанная между собой системной сетью и функционирующая с точки зрения пользователя как единый вычислительный узел. Локальные сети отличаются от кластеров тем, что узлы локальной сети используются индивидуально, в соответствии со своим назначением. В свою очередь кластеры разделяются на Высокоскоростные (High Performance, HP) и Системы Высокой Готовности (High Availability, HA), а также Смешанные Системы.

Высокоскоростные системы предназначены для задач, которые требуют больших вычислительных мощностей: обработка изображений, научные исследования, математическое моделирование и т. д.

Кластеры высокой готовности используются в банковских операциях, электронной коммерции и т д.

Супер ЭВМ – МИМД вычислители Флинну разделяются на ЭВМ с общей памятью (SMP) и с распределенной памятью (MPP).

Вычислительные системы архитектуры: MPP (Massively Parallel Processor, Massive Parallel Processing) - массив (матрица), состоящая из некоторого количества процессорных элементов с локальной памятью (микропроцессоров). Это самые мощные ЭВМ. Сети обмена между процессорными элементами для супер ЭВМ такие же как и для кластеров.

Сети обмена делятся на три типа: шины, статические сети, динамические сети.

Быстродействие сети обмена определяется быстродействием узлов и связей между ними, параметры последних определяются: пропускной способностью непрерывного потока данных и максимальным количеством самых маленьких пакетов, которые можно передать за единицу времени (время передачи пакета нулевой длины, задержка). Некоторые сетевые интерфейсы

Технология Mbyte/s Задержка млсек

Fast Ethernet	12.5	156
Gigabit Ethernet	125	33
Myrinet	245	6
SCI	400	1.5

Сети обмена делятся на три типа: шины, статические сети, динамические сети.

Шины

Система связи ПЭ через коммутирующую общую шину отличаются от других схем простотой, однако ее производительность ограничивается необходимостью обеспечивать в любой момент времени не более одного запроса на передачу информации. Общая шина с центральным коммутатором обеспечивает арбитраж запросов процессоров на передачу информации - выделение один из запросов на доступ к шине и обеспечение монопольного права на владения шиной на все время требуемого обмена. Шины - пропускная способность 100 Мбайт/с для 2 -20 ПЭ (Multimax). Число процессоров в таких системах всегда ограничено.

Статические сети

Статические сети имеют жестко фиксированные соединения, вход и выход зафиксированы без возможности переключения. Наиболее простой топологией сети является линейка – одномерная сетка. Для обеспечения передачи информации между несмежными узлам используются транзитные пересылки. Для цепочки из М узлов наиболее медленная пересылка есть пересылка между конечными ПЭ линейки и она потребует (М-1) транзитных пересылок. Число таких пересылок для любой статической топологии сети считается ее параметром и называется – диаметром сети. Если связать конечные ПЭ линейки, то такая топология - кольцо будет иметь меньший диаметр. Сети могут иметь вид: одномерный линейный массив, двумерное кольцо, звезда, сетка и гексагональный массив, дерево, трехмерное полностью связанное хордовое кольцо, 3 -мерный куб, сети из циклически связанных 3-мерных кубов, D - мерный массив с топологией гиперкуба, тасовка (совершенная, обменная). Недостаток - необходимость маршрутизации транзитных сообщений. По топологии гиперкуба, каждое ПЭ связывается со своим ближайшем соседом в n мерном пространстве. У каждого узла в двумерном гиперкубе имеются два ближайших соседа, в трехмерном - три, в четырехмерном - четыре.

Динамические сети

Динамические сети - сети с возможностью динамического (коммутируемого) соединения узлов сети друг с другом. Особое место занимает полный коммутатор.

Полный коммутатор обеспечивает полную связность каждого узла в сети, причем, обеспечивает независимость (не блокируемость) пересылок.

Недостаток: высокая стоимость аппаратуры и ограниченная размерность. Перечисленные выше однокаскадные сети обмена содержат фиксированное число каскадов или один каскад переключателей. Многокаскадные сети могут быть получены комбинацией некоторых однокаскадных сетей и могут составить конкуренцию полному коммутатору. Например, стандартная сеть Клоша может иметь нечетное число каскадов и строится из сетей меньших размеров. Трехкаскадная сеть Клоша имеет с Н входами и Н выходами имеет р модулей

23. Симметричные мультипроцессоры

Системы данного класса: SMP (Scalable Parallel Processor) состоят из нескольких однородных процессоров и массива общей памяти (разделяемой памяти – shared memory): любой процессор может обращаться к любому элементу памяти. По этой схеме построены 2,4 процессорные SMP сервера на базе процессоров Intel, HP и т. д., причем процессоры подключены к памяти с помощью общей шины. Системы с большим числом процессоров (но не более 32) подключаются к общей памяти, разделенной на блоки, через не блокирующийся полный коммутатор: crossbar. Любой процессор системы получает данное по произвольному адресу памяти за одинаковое время, такая структура памяти называется: UMA - Uniform Memory Access (Architecture). Пример:HP-9000. Дальнейшее масштабирование (увеличение числа процессоров системы) SMP систем обеспечивается переходом к архитектуре памяти: NUMA - Non Uniform Memory Access. По схеме, называемой, этой иногда, кластеризацией SMP, соответствующие блоки памяти двух (или более) серверов соединяются кольцевой связью, обычно по GCI интерфейсу. При запросе данного, расположенного вне локального с сервере диапазона адресов, это данное по кольцевой связи переписывается дублируется в соответствующий блок локальной памяти, ту часть его, которая специально отводится для буферизации глобальных данных и из этого буфера поставляется потребителю. Эта буферизация прозрачна (невидима) пользователю, для которого вся память кластера имеет сквозную нумерацию, и время выборки данных, не локальных в сервере, будет равно времени выборки локальных данных при повторных обращениях к глобальному данному, когда оно уже переписано в буфер. Данный аппарат буферизации есть типичная схема кэш памяти. Так как к данным возможно обращение из любого процессора кластера, то буферизация, размножение данных требует обеспечение их когерентности. Когерентность данных состоит в том, что при изменении данного все его потребители должны получать это значение. Проблема когерентности усложняется дублированием данных еще и в процессорных кэшах системы. Системы, в которых обеспечена когерентность данных, буферизуемых в кэшах, называются кэш когерентными (cc-cache coherent), а архитектура памяти описываемого кластера: cc- NUMA (cache coherent Non Uniform Memory Access). Классической архитектурой принято считать систему SPP1000.

24. Статический и динамический способы образования параллельных процессов.

"Процесс - группа ячеек памяти, содержимое которых меняется по определенным правилам. Эти правила описываются программой, которую интерпретирует процессор." /Цикритзис Д./.

Вычислительный процесс можно рассматривать как последовательность команд ЭВМ, которая работает с данными ей ресурсами. (Задача, задание, процесс, нить, TASK). Так, на микропроцессоре могут работать одновременно несколько независимых процессов: счет вычислительной задачи, редактирование текстов и т.д. Далее рассматриваются вычислительные процессы, одновременно выполняющиеся на нескольких процессорах для решения общей задачи. Работа такой задачи может начинаться с порождения главного процесса, который при работе может порождать другие процессы динамически. Эти процессы могут работать параллельно с главной и также порождать другие процессы.

Другим способом порождения процессов является статический способ, когда производится одновременная инициализация на всех процессорах одинаковых процессов (SPMD – Single Program Multiple Data). Эти процессы опрашивают состояние решающего поля и настраиваются на выполнение своей части вычислений.(Они могут узнать: сколько их порождено, свои уникальные, внутренние имена, и т.д.)

Процессы, работающие на равных правах (не вызовы процедур и не процессы, связанные понятиями "главная-подчиненная"), иногда называемые сопроцессами, могут выполняться параллельно и при этом общаться друг с другом - не часто (слабо связанные процессы).

25. Двоичные и общие семафоры

Процессы для своей работы могут использовать логические и физические ресурсы вычислительной системы и ее окружения, причем ресурсы могут быть: поделены между процессами и закреплены постоянно (на все время работы процесса) или использованы всеми или некоторыми процессами по-очереди. Некоторые ресурсы могут быть общими и допускать параллельное обслуживание процессов. Ресурс, который допускает обслуживание только одного процесса в текущее время, называется критическим ресурсом.

Участки программы процесса, в которых происходит обращение к критическим ресурсам, называются критическими участками. Такие участки должны быть выполнены в режиме "взаимного исключения", т.е. в каждый момент времени не более чем один процесс может быть занят выполнением своего, критического относительно некоторого ресурса, участка. Проблема критического участка - программирование работы критических участков в режиме взаимного исключения решается с помощью семафоров. Общий семафор - это целочисленная переменная, над которой разрешены только две неделимые операции P и V. Аргументом у этих операций является семафор. Определять семантику этих операций можно только для семафоров - положительных чисел, или включать и отрицательный диапазон. Операции могут быть определены так:

P(S) - декремент и анализ семафора

```
S := S-1
IF (S < 0) THEN  <блокировка текущего процесса>
ENDIF
```

V(S) - инкремент семафора

```
S := S+1
IF S <= 0 THEN <запустить блокированный  процесс>
ENDIF
```

P и V операции неделимы: в каждый момент только один процесс может выполнять P или V операцию над данным семафором.

Если семафор используется как счетчик ресурсов, то:

S >= 1 - есть некоторое количество (S) доступных ресурсов,

S = 0 - нет доступного ресурса,

S < 0 - один или несколько (S) процессов находятся в очереди к ресурсу.

Вводится, также, операция инициализации семафора (присвоение начального значения).

Общий семафор - избыточный, т.к. его можно реализовать через двоичные семафоры, которые принимают только два значения 0,1.

Семафоры позволяют:

- синхронизировать работу процессов,
- управлять распределением ресурсом (использованием положительного диапазона значений семафора как счетчика ресурсов) ,
- организовывать очередь блокированных процессов (отрицательные значения семафора показывают число блокированных процессов).

В системах программирования вводится специальный тип данных, определяющий структуру семафора, например, SEMAPHOR,SIGNAL,GETA.

Использование семафоров

```
begin  integer S; S:=1;
 parbegin
 task1: begin
            do while (true)
            P(S);
            <критический участок 1>;
            V(S);
            <обычный участок>;
            enddo
        end;
 task2: begin
            do while (true)
            P(S);
```

```
            <критический участок 2>;
            V(S);
            <обычный участок>;
            enddo
        end
  parend
end
```

26. Назначение системы MPI

Главные цели создания системы параллельного программирования:

Создать интерфейс прикладного программирования для MPP систем;

Обеспечить возможность эффективных коммуникаций для коммуникации точка-точка, коллективных операций, группы процессов.

Допускать удобное сопряжение с языками C, Fortran 77, Fortran 90 и C++;

Простой способ создания процессов для модели SPMD (одна программа используется для обработки разных данных на разных процессорах);

Основные понятия языка

Группа - упорядоченное (от 0 до ранга группы) множество идентификаторов процессов Группы служат для указания адресата при посылке сообщений (процесс-адресат специфицируется своим номером в группе), определяют исполнителей коллективных операций. Являются мощным средством функционального распараллеливания - позволяют разделить группу процессов на несколько подгрупп, каждая из которых должна выполнять свою параллельную процедуру. При этом существенно упрощается проблема адресации при использовании параллельных процедур.

Контекст - область "видимости" для сообщений, аналогичное области видимости переменных в случае вложенных вызовов процедур. Сообщения, посланные в некотором контексте, могут быть приняты только в этом же контексте. Контексты - также важные средства поддержки параллельных процедур.

Коммуникаторы - позволяют ограничить область видимости (жизни, определения) сообщений рамками некоторой группы процессов, т.е. могут рассматриваться как пара - группа и контекст. Кроме того, они служат и для целей оптимизации, храня необходимые для этого дополнительные объекты.

Имеются предопределенные коммуникаторы (точнее, создаваемые при инициализации MPI-системы): * MPI_COMM_ALL - все процессы и операции над группами (локальные, без обмена сообщениями), например, Дай размер группы. MPI_GROUP_SIZE(IN group, OUT size) Дай номер в группе обратившегося процесса. MPI_GROUP_RANK(IN group, OUT rank)

Основные операции - send, receive

Операции могут быть блокирующими и неблокирующими.

В операции send задается:

- адрес буфера в памяти;
- количество посылаемых элементов;
- тип данных каждого элемента;
- номер процесса-адресата в его группе;
- тег сообщения;
- коммуникатор.

MPI_SEND(IN start, IN count, IN datatype, IN dest, IN tag, IN comm) Типы данных - свои в MPI, но имеется соответствие между ними и типами Fortran и C.

В операции receive задается:

- адрес буфера в памяти;
- количество принимаемых элементов;
- тип данных каждого элемента;
- номер процесса-отправителя в его группе;
- тег сообщения;
- коммуникатор;
- ссылка на объект-статус, содержащий необходимую информацию о состоянии и результате операции.

Имеется возможность указать "любой отправитель" и "любой тег".

Имеется три режима коммуникаций - стандартный, режим готовности и синхронный.

В стандартном режиме последовательность выдачи операций send и receive произвольна, операция send завершается тогда, когда сообщение изъято из буфера и он уже может использоваться процессом.

В режиме готовности операция send может быть выдана только после выдачи соответствующей операции receive, иначе программа считается ошибочной и результат ее работы неопределен. В синхронном режиме последовательность выдачи операций произвольна, но операция send завершается только после выдачи и начала выполнения операции receive. Во всех трех режимах операция receive завершается после получения сообщения в заданный пользователем буфер приема.

Неблокирующие операции не приостанавливают процесс до своего завершения , а возвращают ссылку на коммуникационный объект, позволяющий опрашивать состояние операции или дожидаться ее окончания. Имеются операции проверки поступающих процессу сообщений, без чтения их в буфер (например, для определения длины сообщения и запроса затем памяти под него).

27. Распараллеливание последовательных программ. (Метод гиперплоскостей)

Метод гиперплоскостей применим только к многомерным циклам. В пространстве итераций ищется прямая (плоскость), на которой возможно параллельное асинхронное выполнение тела цикла, причем в отличие от метода

координат, эта прямая (плоскость) может иметь наклон по отношению к осям координат. Цикл вида:

```
      DO  I = 2,N
      DO  J = 2,M
    A(I,J) = ( A(I-1,J) + A(I,J-1) ) * 0.5
```

методом координат не векторизуется. Действительно, при фиксированном значении переменной *I* *(I = i)* значение, вычисленное в точке *(i,j)* *пространства итераций, зависит от результата вычислений в предыдущей точке (i,j-1)*, так что параллельное выполнение тела цикла по переменной *J* невозможно. Аналогично нельзя проводить параллельные вычисления по переменной *I*.

Однако можно заметить, что результат будет также правильным, если вычисления проводить в следующем порядке:

на 1-м шаге - в точке (2,2),
на 2-м шаге - в точках (3,2) и (2,3),
на 3-м шаге - в точках (4,2), (3,3) и (2,4),
на 4-м шаге - в точках (5,2), (4,3), (3,4) и (2,5) и т.д.

Вычисления в указанных точках для каждого шага, начиная со второго, можно проводить параллельно и асинхронно. Перечисленные кортежи точек лежат на параллельных прямых вида *I+J=K*, а именно: на первом шаге - на прямой *I+J=4*, на втором - *I+J=5*, на третьем шаге - *I+J=6* и т.д., на последнем *((N-2)+(M-2)+1)* - ом шаге - на прямой *I+J=M+N*.

Для приведенного примера множество точек, в которых возможно параллельное выполнение, является однопараметрическим (прямая) и определяется из решения уравнения *I+J=K*. Цикл (5) может быть преобразован к виду: DO 5 K = 4,M+N

```
      Тн = MMAX(2,K-N)
      Тк = MMIN(M,K-2)
      DO 5  T = Тн,Тк : PAR
      I = T
      J = K - T
    5 A(I,J) = ( A(I-1,J) + A(I,J-1) ) * 0.5
```

Функция *MMAX(X,Y)* выбирает ближайшее целое, большее или равное максимуму из чисел *X* и *Y*, а функция *MMIN(X,Y)* - ближайшее целое, меньшее или равное минимуму из *X* и *Y*.

Внутренний цикл по переменной *T* может выполняться параллельно для всех значений *T*. Границы изменения переменной цикла *T* меняются при переходе с одной прямой (гиперплоскости) на другую, поэтому их необходимо перевычислять во внешнем цикле. Число итераций внутреннего цикла, то есть потенциальная длина векторной операции, меняется с изменением

28. Ограничения на распараллеливание циклов.

Распараллеливание циклов возможно, если все итерации цикла независимы. Тело цикла не должны содержать:

- операторов перехода
- операторов ввода-вывода

Индексные выражения не должны иметь индекс в индексе A(B(C))

29. Алгоритмы преобразования программ методом координат

Метод координат:

- позволяет определить возможность выполнения цикла в синхронном режиме(для архитектуры SIMD);
- содержит алгоритмы преобразования тела цикла к синхронному виду. Например, по Лэмпорту, цикл:

```
      DO 24 I=2,M
      DO 24 J=2,N
21    A(I,J) = B(I,J)+C(I)
22    C(I) = B(I-1,J)
23    B(I,J) = A(I+1,J) ** 2
24    CONTINUE
```

преобразуется в цикл:

```
      DO 24 J=2,N
      DO 24 SIM FOR ALL I i:2<=i<=M C SIM - SI Multeneusly
(одновременно)
      TEMP(I) = A(I+1,J)
21  A(I,J) = B(I,J)+C(I)
23    B(I,J) = TEMP(I) ** 2
22  C(I) = B(I-1,J)
24    CONTINUE
```

Примеры векторизации

Исходные тела циклов	Преобразованные тела циклов
A(I) = B(I) C(I) = A(I+1)	C(I) = A(I+1) A(I) = B(I)
A(I) = B(I) C(I) = A(I) + A(I+1)	D(I) = A(I) A(I) = B(I) C(I) = A(I) + D(I+1)

30. Синхронный и асинхронный способы передачи сообщений

В MPI имеется три режима коммуникаций - стандартный, режим готовности и синхронный.

В стандартном режиме последовательность выдачи операций send и receive произвольна, операция send завершается тогда, когда сообщение изъято из буфера и он уже может использоваться процессом.

В режиме готовности операция send может быть выдана только после выдачи соответствующей операции receive, иначе программа считается ошибочной и результат ее работы неопределен. В синхронном режиме последовательность выдачи операций произвольна, но операция send завершается только после выдачи и начала выполнения операции receive. Во всех трех режимах операция receive завершается после получения сообщения в заданный пользователем буфер приема.

Неблокирующие операции не приостанавливают процесс до своего завершения , а возвращают ссылку на коммуникационный объект, позволяющий опрашивать состояние операции или дожидаться ее окончания. Имеются операции проверки поступающих процессу сообщений, без чтения их в буфер (например, для определения длины сообщения и запроса затем памяти под него).

31. Параллельное выполнение цикла вида

DO i=2,N A(I) =(B(i)+C(i))/A(i+const) ENDDO.

Если const = 0, то все итерации цикла независимы и токой цикл может быть выполнен на любой многопроцессорной ЭВМ, влючая Иллиак-4. (каждый виток цикла выполняется на отдельном процессоре)

Если const > 0 (пусть =1) то при параллельное выполнение цикла на ЭВМ класса МКМД без дополнительных мер по синхронизации работы процессоров невозможно. Например, пусть N=3 и два процессора вычисляют параллельно эти итерации, тогда, первый процессор вычисляет A2 по B2, C2, A3, а второй, A3 по B2, C2, A4 . При отсутствии синхронизации может случиться ситуация, при которой второй процессор завершит свою работу до начала работы первого. Тогда первый процессор для вычислений будет использовать A3, которое обновил второй процессор, что неверно, ибо здесь нужно “старое” значение A3. Однако, этот цикл выполняется параллельно на ЭВМ ОКМД (SIMD) так как там этот цикл может быть выполнен такими командами:

1. Считать Bi в сумматоры каждого из n АЛУ.
2. Сложить Ci со своим содержимом сумматора.
3. Разделить содержимое каждого i-сумматора на Ai+1.
4. Записать содержимое i- сумматоров в Ai.

Из за того, что выборка из памяти и запись в память производится синхронно (одновременно), то работа цикла – корректна.

Если const < 0 то параллельное выполнение цикла невозможно, ибо для выполнения очередной итерации цикла необходимы результаты работы предыдущей (рекурсия). Однако известны приемы преобразования такого рода циклов к виду, допускающие параллельное выполнение.

24. Статический и динамический способы образования параллельных процессов. "Процесс - группа ячеек памяти, содержимое которых меняется по определенным правилам. Эти правила описываются программой, которую интерпретирует процессор" /Цикритзис Д./.

32. Распараллеливание алгоритмов сложения методом редукции.

Рекурсия - последовательность вычислений, при котором, значение самого последнего терма в последовательности зависит от одного или несколько ранее вычисленных термов. Пусть группа вычислений может производиться параллельно, использую результаты вычислений, выполненных на предыдущих этапах (полученных в виде начальных данных). Тогда, каждая группа вычислений называется "ярусом" параллельной формы, число групп - "высотой", максимальное число операций в группе "шириной" параллельной формы. Один и тот же алгоритм может иметь несколько представлений в виде параллельных форм, различающиеся как шириной, так и высотой. Редукционный алгоритм сдваивания для суммирования чисел с получением частных сумм может иметь вид:

Данные A1 A2 A3 A4 A5 A6 A7 A8
Ярус 1 A1+A2 A3+A4 A5+A6 A7+A8
Ярус 2 A12+A3 A12+A34 A56+A7 A56+A78
Ярус 3 A1234+A5 A1234+A56 A1234+A567 A1234+A5678

Высота параллельной формы равна трем, ширина - четырем, причем загрузка вычислителей (четырех) полная. В данном алгоритме производится вычисления пяти "лишних" чисел по сравнению с последовательным алгоритмом сложения восьми чисел.

33. Метод распараллеливания алгоритма общей рекурсии 1-го порядка.

Редукция - упрощение, в биологии уменьшение размера органа вплоть до его полного исчезновения. Циклическая редукция - алгоритмы численного анализа для распараллеливания последовательных алгоритмов, основанный на последовательном, циклическом применении параллельных вычислений, число которых на каждом этапе уменьшается (делится пополам).

Общей линейной рекурсией первого порядка называется система уравнений вида:X1 = D1

X2 = X1 * A2 + D2
Xi = Xi-1 * Ai + Di
Xn = Xn-1 * An + Dn

в общем виде: Xi = Xi-1 * Ai + Di, i = 2,3,...n, X1 = D1

Последовательный алгоритм вычислений может быть записан так:

```
X(1) = A(1) + D(1)
DO i = 2,n
  X(i) = X(i-1) * A(i) + D(i)
```

ENDDO

Рекурсивная зависимость итераций цикла не позволяет ускорить вычисления за счет параллельной работы оборудования. Преобразуем данный алгоритм в параллельный методом циклической редукции. Рассмотрим два соседних уравнения:

Xi-1 = Xi-2 * Ai-1 + Di-1

Xi = Xi-1 * Ai + Di

и подставив первое во второе, получаем:

Xi = (Xi-2 * Ai-1 + Di-1) * Ai + Di = Xi-2 * A1i + D1i , где

A1i = Ai * Ai-1 ,

D1i = Ai * Di-1 + Di

Тогда, проведя эту операцию для всей системы уравнений, получим систему уравнений порядка n/2. Если повторить процедуру l раз (если n = 2**l), то в результате получается значение: Xn = Dnl. Для получения полного вектора X необходимо модифицировать алгоритм, например, по аналогии с алгоритмами суммирования.

Очевидно, что вычисления Aji и Dji можно проводить параллельно методом каскадных сумм с сохранением частных сумм. Приведенные уравнения для уровня i имеют вид:

Xi = Ali * Xi-2**l + Dli , где l = 0,1,..,log2n , i = 1,2,..,n

Ali = Al-1i * Al-1(i-2**l-1)

Dli = Al-1i * Dl-1(i-2**l-1) + Dl-1i

Начальные данные: A0i = Ai, D0i = Di

Если индекс i у любого Ali, Dli и Xi попадает вне диапазона 1 <= i <= n , то он должен быть приравнен к нулю. Тогда , при l = log2n в уравнениях: Xi = Ali * Xi-2**l + Dli индекс Xi-2**l = Xi-n находится вне диапазона, и, следовательно, решением системы уравнений будет:

вектор: Xi = Dli, Векторная нотация Хокни для данного алгоритма: X = D

```
DO L = 1,LOG2(N)
   X = A * SHIFTR(X,2**(L-1)) + X
   A = A * SHIFTR(A,2**(L-1))
ENDDO
```

24. Системы счисления.

Подмножество вещественных чисел, которое может быть представлено в ЭВМ в форме чисел с плавающей запятой, принято обозначать буквой F и определять его элементы для конкретной архитектуры - "машинные числа", (по Форсайту и др.) четырмя целочисленными параметрами: базой b, точностью t и интервалом значений показателя [L,U]. Множество F содержит число нуль и все f числа вида: f = (+/-).d1d2...dt * b**e, где e называется показателем, число .d1d2...dt = (d1/b+ ....+dt/(b**t)) - дробной частью - мантиссой, причем: 0<=di<b, L<=e<=U.

Каноническая или нормализованная форма F определяется дополнительным соотношением d1 =/= 0 ; это условие позволяет устранить неоднозначность представления одинаковых чисел, дает наивысшую возможную точность представления чисел. Особенности F:

- для каждого ненулевого f верно: m<=|f|<=M, где m = b**(L-1),
  M = (b**U) * (1-b**(-t));
- множество F конечно и содержит 2*(b-1)*(b**(t-1))*(U-L+1)+1 чисел, которые отстоят друг от друга на числовой оси на неравные промежутки.

35. Определить минимальное значение числа с плавающей запятой

- для каждого ненулевого f верно: m<=|f|<=M, где m = b**(L-1),
  M = (b**U) * (1-b**(-t));

36. Определить количество элементов чисел с плавающей запятой.

- множество F конечно и содержит 2*(b-1)*(b**(t-1))*(U-L+1)+1 чисел, которые отстоят друг от друга на числовой оси на неравные промежутки.

37. Машинный эпсилон, определение разрядной сетки ЭВМ.

Точность плавающей арифметики можно характеризовать посредством машинного эпсилона. Максимальное число E такое, что 1.+ E = 1. является мерой точности представления чисел на данной ЭВМ (машинное эпсилон). Грубая схема вычисления эпсилона:

```
      EPS = 1.0
    1 EPS = 0.5 * EPS
      EPS1 = EPS + 1.0
      IF (EPS1 .GT. 1.0) GO TO 1   >
```

Задача. Написать программу, определяющую количество разрядов, используемых для представления мантиссы чисел с плавающей запятой. (Пусть на испытываемой ЭВМ мантисса числа хранится в нормализованном виде 1A2A3...An).

38. Источники погрешности при вычислениях на параллельных системах.

В общем случае, арифметические операции над элементами дискретного подмножества вещественных чисел F не корректны.

Результат арифметических операций чисел с плавающей запятой может:

- иметь абсолютное значение, больше M (максимального числа) - машинное переполнение;
- иметь ненулевое значение, меньшее m (минимального числа) по абсолютной величине - машинный нуль;
- иметь значение в диапазоне [m:M] и тем не не менее не принадлежать множеству F (произведение двух чисел из F, как правило, записывается посредством 2t либо 2t-1 значащих цифр);

Поэтому, на множестве чисел с плавающей запятой определяются и "плавающие" арифметические операции, за результаты которых, если они не выходит за границы множества F, принимается ближайшие по значению элементы F. Примеры из четырехразрядной десятичной арифметики по Н. Вирту.

А) Пусть x=9.900 y=1.000 z=-0.999 и тогда:

1 (x+y)+z = 9.910

2 x+(y+z) = 9.901

В) Пусть x=1100. y=-5.000 z=5.001 и тогда:

1 (x*y)+(x*z) = 1.000

2 x*(y+z) = 1.100

Здесь операции + и * - плавающие машинные операции. Такие 'численные' процессы называют иногда 'неточными', здесь нарушаются ассоциативный и дистрибутивный законы арифметики..

39. Оценить полную ошибку для суммирования положительных чисел.

Пример расчета полной ошибки для суммирования положительных чисел

Формула полной ошибки для суммирования положительных чисел Ai(i=1,..,n) имеет вид: Ds = A1*da1 + A2*da2 +...+ An*dan + d1*(A1+A2) +..+ d(n-1)*(A1+..+An) + dn , где

dai - относительные ошибки представления чисел в ЭВМ, a di - относительные ошибки округления чисел при каждой следующей операции сложения. Пусть: все dai = da и di = d , a Ks = A1+A2+..+An, тогда: Ds = da*Ks + d*[(n-1)*A1+(n-1)*A2 +...+ 2*A(n-1) + An]

Очевидно, что наибольший "вклад" в сумму ошибок вносят числа, суммируемые вначале. Следовательно, если суммируемые положительные числа упорядочить по возрастанию, максимально возможная ошибка суммы будет минимальной. Изменяя порядок суммирования чисел можно получать различные результаты. Но если даже слагаемые отличаются друг от друга незначительно, на точность результата может оказать влияние способ суммирования. Пусть суммируются 15 положительных чисел, тогда ошибка результата: Ds = da*Ks + d*(14*A1+14*A2+13*A3+....+2*A14+A15).

Слагаемое da*Ks не зависит от способа суммирования, и далее не учитывается. Пусть слагаемые имеют вид: Ai = A0+ei, где i=1,...,15, тогда: Dss = 199*(A0+em)*d, где em = max(ei), d - ошибка округления при выполнении арифметической операции сложения.

Если провести суммирование этих чисел по группам (три группы по четыре числа и одна группа из трех чисел), то ошибки частных сумм имеют вид:

Ds1 = d*(3*A1+3*A2+2*A3+A4) <= 9*d*(A0+em)

Ds2 = d*(3*A5+3*A6+2*A7+A8) <= 9*d*(A0+em)

Ds3 = d*(3*A9+3*A10+2*A11+A12) <= 9*d*(A0+em)

Ds4 = d*(3*A13+2*A14+A15) <= 5*d*(A0+em)

а полная оценка ошибок округления будет Ds <= 32*d*(A0+em), что меньше Dss. Итак суммирование по группам дает меньшую ошибку результата.

Например, разделив процесс суммирования массива положительных чисел на параллельные процессы, и затем получив сумму частных сумм, можно получить результат, отличный от последовательного суммирования в одном процесс.

40. Определить .....

Эффект от буферизации можно определить среднему времени выборки: t = t2*p+ t1*(1-p), где t1 - среднее время доступа к данным основной памяти, t2 - среднее время доступа к данным из буфера (t2<t1), p - вероятность наличия данного в буфере.

41. Привести пример программы, вызывающей перезагрузку кэша с прямым распределением.

В кэше с прямым отображением адреса кэша для передаваемых объектов данных являются функцией полного виртуального адреса объектов. Для кэша объемом в 1 Мбайт адреса процессорного кэша вычисляются в процессе работы по формуле: cache_address=MOD(virtual_address,2**20) MOD - функция, возвращающая остаток от деления virtual_address на 2**20 (2**20=1048576=1 Мбайт). Таким образом, адрес кэша объекта данных - это младшие 20 бит его виртуального адреса. Старшие разряды адреса – теги храняться в специальной памяти – памяти тэгов. Такая схема адресации может стать причиной перезагрузки кэша (кэш трэшинг), если последовательно используемые процессором данные расположены в оперативной памяти по адресам, кратным размеру памяти кэша. Пусть 1- М-байтный кэш процессора прямо отображается на виртуальную память. Это означает, что обращение к памяти с шагом в 1 - М-байт "гарантирует" перезапись строк кэша [строка кэша - блок данных, читаемые из или записываемые в (или из) память за одно обращение; 32 байта для кэша процессора]. Пусть имеется программа:

```
REAL *8 A(65536),B(65536),C(65536)
DO I = 1,N
 A(I) = B(I) + C(I)
ENDDO
```

Каждый массив имеет размер - точно 1/2 М-байта. Предположим, что они загружены в память без разрывов. Тогда, все строки кэша для С в точности накладываются на строки кэша для массива А. Так что, процессор, выполняющий 1 итерацию цикла, будет загружать строку кэша массива В, строку кэша массива С, вычислит B(I) + C(I) , а затем загрузит строку кэша А (которая приведет к перезаписи строки кэша С). На следующей итерации строки кэша для С и А должны быть выбраны снова, так что не будет повторного использования данных из кэша без перезаписи. Добавка некоторого смещения к размерности массива исключает эту проблему. Таким образом, следующий код будет выполняться значительно быстрее:

```
REAL *8 A(65536+8),B(65536+8),C(65536+8)
DO I = 1,N
```

```
  A(I) = B(I) + C(I)
ENDDO
```

Такое изменение размерности гарантирует, что строки кэша, которые требуются в одной итерации, не будут записываться друг на друга.

Приложение: Стандартные префиксы Международной системы измерений СИ

префикс	двоичная система	десятичная система	
кило-К	1024**1 = 2**10 = 1024	1000**1 = 10**3	тысяча
мега-М	1024**2 = 2**20 = 1 048 576	1000**2 = 10**6	миллион
гига-Г	1024**3 = 2**30 = 1 073 741 824	1000**3 = 10**9	миллиард, биллион
тера-Т	1024**4 = 2**40 = 1 099 551 627 776	1000**4 = 10**12	триллион
пета-П	1024**5 = 2**50 = 1 125 899 906 842 624	1000**5 = 10**15	квадриллион
екза-	1024**6 = 2**60 = ...	1000**6 = 10**18	квинтиллион
зетта-	1024**7 = 2**70	1000**7 = 10**21	сексиллион
йотта-	1024**8 = 2**80	1000**8 = 10**24	септииллион

Префиксы, используемые при умножении на отрицательную степень 1000

милли-	1000** -1	= 10** -3	= 0.001
микро-	1000** -2	= 10** -6	= 0.000001
нано-	1000** -3	= 10** -9	= 0.000000001
пико-	1000** -4	= 10** -12	= .....
фемто-	1000** -5	= 10** -15	
атто-	1000** -6	=	
зенто-	1000** -7		
йокто-	1000** -8		

В АНГЛИИ:

Биллион = миллиону миллионов = 1 с 12 нулями

Триллион = миллиону биллионов = 1 с 18 нулями

Квадриллион = миллиону триллионов = 1 с 24 нулями

Квинтиллион = миллиону квадриллионов = 1 с 30 нулями

Секстиллион = миллиону квинтиллионов = 1 с 36 нулями

тактовая частота в МГц	продолжительность цикла в нс
60	15
100	10
133	7.5
166	6

Особенности вычисления выражений в Фортране 90 (пункты стандарта).

7.1.7. Выполнение операций

"При вычислении выражения A+B, где A,B - массивы, процессор может выполнять по-элементные операции сложения в любом порядке"

7.1.7.1. Вычисление операндов

"Процессор не обязательно вычисляет все операнды выражения или полностью каждый операнд, если значения можно определить без этого. Так в выражении: X .GT. Y .OR. L(Z) не надо вычислять функцию L(Z), если X больше Y."

7.1.7.2. Целостность скобок

"Для выражения: A+(B+C) процессор на должен вычислять математически эквивалентное выражения: (A+B)+C."

7.1.7.3. Выполнение числовых встроенных операций

"... процессор может вычислять любое математически эквивалентное выражение при условии, что целостность скобок не нарушается."

В стандарте языка Фортран 90 приводятся допустимые и не допустимые альтернативных формы преобразования выражений:

Для целых i,j; вещественных и комплексных a,b,c; произвольных x,y,z

разрешаются преобразования:	и запрещаются:
x+y -> y+x	i/2 -> 0.5 *i
x*y -> y*x	x*i/j -> x*(i/j)
-x+y -> y-x	(x+y)+z -> (x+y+z)
x+z+y -> y+(x+z)	i/j/a -> i/(j*a)
x*a/z -> x*(a/z)	(x+y)+z -> x+(y+z)
a/b/c -> a/(b*c)	(x*y)-(x*z) -> x*(y-z)
a/5.0 -> 0.2*a	x*(y-z) -> x*y-x*z