Методы параллельных вычислений

1. **Синхронизация. Два вида синхронизации.**

Синхронизация (synchronization) - это механизм, управляющий порядком выполнения задач. При совместном использовании данных, а также когда задачи взаимодействуют друг с другом или конкурируют между собой, нужны два вида синхронизации. Синхронизация взаимодействия (cooperation synchronization) между задачами A и B требуется, когда задача A должна ожидать, пока задача B закончит выполнение определенных действий, прежде чем задача A сможет продолжить свою работу. Синхронизация конкуренции (competition synchronization) между двумя задачами необходима, когда обе задачи нуждаются в использовании некоторого ресурса, который невозможно использовать одновременно. В частности, если задача A нуждается в получении доступа к совместно используемой ячейке x в то время, когда эта ячейка находится в распоряжении задачи B, то задача A должна ожидать, пока задача B не закончит обработку ячейки x, независимо от того, что именно представляет собой эта обработка.

Общий подход к обеспечению взаимно исключающего доступа к совместно используемому ресурсу заключается в том, что этот ресурс рассматривается как сущность, которая одновременно может принадлежать только одной задаче. Чтобы получить право владения совместно используемым ресурсом, задача должна запросить его. Завершив свое выполнение, задача должна освободить ресурс, чтобы он стал доступен другим задачам.

Три метода обеспечения взаимно исключающего доступа к совместно используемым ресурсам: семафоры, мониторы, передача сообщений.

Механизмы синхронизации должны иметь возможность задерживать выполнение задач. Синхронизация устанавливает порядок выполнения задач, определяемый этими задержками. Программа, называемая **планировщиком** (scheduler), управляет распределением процессоров между задачами. Если не происходит никаких прерываний и все задачи имеют одинаковый приоритет, то планировщик просто предоставляет каждой задаче некоторый отрезок времени. Когда подходит очередь какой-либо задачи, планировщик позволяет процессору выполнять ее в течение некоторого времени.

2. **Императивное программирование в отличии от декларативного программирования. Асинхронное параллельное программирование?**
   Кроме императивных (явный параллелизм) программ имеются

декларативные программы — функциональные и логические (неявный параллелизм).
Асинхронное программирование (условие → действие)
Векторные и конвейерные вычисления

3. **Состав однопроцессорной машины.**

Современная однопроцессорная машина состоит из нескольких компонентов: центрального процессорного устройства (ЦПУ), первичной памяти, одного или нескольких уровней *кэш*-памяти (*кэш*), вторичной (дисковой) памяти и набора периферийных устройств (*дисплей*, клавиатура, *мышь*, *модем*, CD, принтер и т.д.). Основными компонентами для выполнения программ являются ЦПУ, *кэш* и *память*.

Процессор выбирает инструкции из памяти, декодирует их и выполняет. Он содержит *управляющее устройство* (УУ), *арифметико-логическое устройство* (*АЛУ*) и регистры. УУ вырабатывает сигналы, *управляющие* действиями *АЛУ*, системой памяти и внешними устройствами. *АЛУ* выполняет арифметические и логические инструкции, определяемые набором инструкций процессора. В регистрах хранятся инструкции, данные и состояние машины (включая *счетчик команд*).

4. **Состав мультипроцессорной машины. Виды организации памяти.**

Мультикомпьютеры с распределенной памятью

В мультикомпьютерах с распределенной памятью существуют соединительная *сеть*, но каждый *процессор* имеет собственную *память*. Соединительная *сеть* поддерживает передачу сообщений. Мультикомпьютеры (многопроцессорные системы с распределенной памятью) не обеспечивают общий *доступ* ко всей имеющейся в системах памяти. Каждый *процессор* системы может использовать только свою локальную *память*, в то время как для доступа к данным, располагаемых на других процессорах, необходимо использовать интерфейсы передачи сообщений (например, стандарт *MPI*). Данный подход используется при построении двух важных типов многопроцессорных вычислительных систем - массивно-параллельных систем (massively *parallel processor* or *MPP*) и кластеров (clusters).

Мультикомпьютер (многомашинная система) – *мультипроцессор* с распределенной памятью, в котором процессоры и *сеть* расположены физически близко (в одном помещении). Также называют тесно связанной машинной. Она одновременно используется одним или небольшим числом приложений; каждое *приложение* задействует выделенный набор процессоров. Соединительная *сеть* с большой пропускной способностью предоставляет высокоскоростной *путь* связи между процессорами.

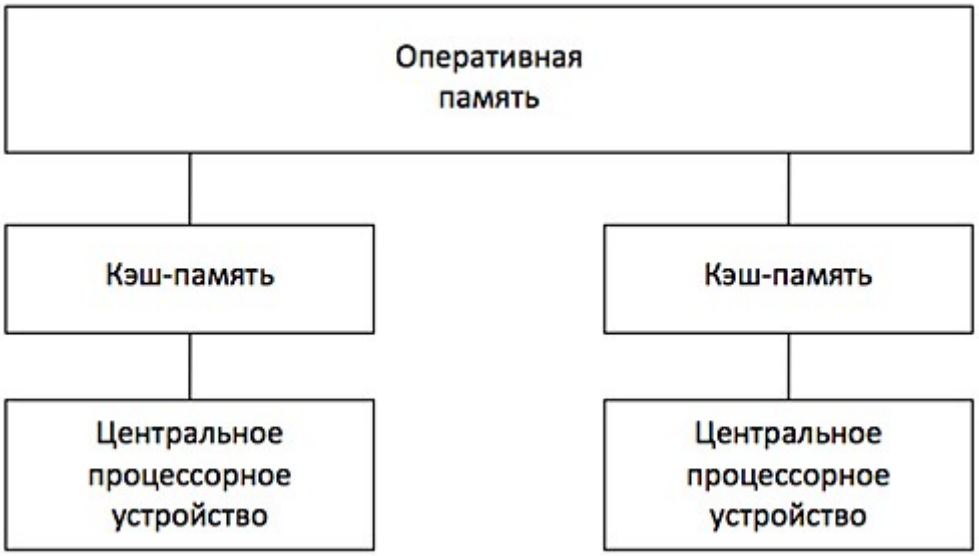
Сетевая система – это многомашинная система с распределенной памятью, связаны с помощью локальной сети или глобальной сети *Internet* (слабо связанные мультикомпьютеры). Здесь процессоры взаимодействуют также с помощью передачи сообщений, но время их доставки больше, чем в многомашинных системах, и в сети больше конфликтов. С другой стороны, сетевая система строится на основе обычных рабочих станций и сетей, тогда как в многомашинной системе

часто есть специализированные компоненты, особенно у связующей сети.

Под *кластером* обычно понимается множество отдельных компьютеров, объединенных в *сеть*, для которых при помощи специальных аппаратно-программных средств обеспечивается возможность унифицированного управления, надежного функционирования и эффективного использования. Кластеры могут быть образованы на базе уже существующих у потребителей отдельных компьютеров, либо же сконструированы из типовых компьютерных элементов, что обычно не требует значительных финансовых затрат. Применение кластеров может также в некоторой степени снизить проблемы, связанные с разработкой параллельных алгоритмов и программ, поскольку повышение вычислительной мощности отдельных процессоров позволяет строить кластеры из сравнительно небольшого количества (несколько десятков) отдельных компьютеров (lowly parallel *processing*). Это приводит к тому, что для параллельного выполнения в алгоритмах решения вычислительных задач достаточно выделять только крупные независимые части расчетов (coarse *granularity*), что, в свою *очередь*, снижает сложность построения параллельных методов вычислений и уменьшает потоки передаваемых данных между компьютерами кластера. Вместе с этим следует отметить, что организация взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам, что накладывает дополнительные ограничения на тип разрабатываемых параллельных алгоритмов и программ.

Мультипроцессор с разделяемой памятью

В мультипроцессоре и в многоядерной системе исполнительные устройства (процессоры и ядра процессоров) имеют *доступ* к разделяемой оперативной памяти. Процессоры совместно используют оперативную *память*.



У каждого процессора есть собственный *кэш*. Если два процессора ссылаются на разные области памяти, их содержимое можно безопасно поместить в *кэш* каждого из них. Проблема возникает, когда два процессора обращаются к одной области памяти. Если оба процессора только считывают данные, в *кэш* каждого из них можно поместить копию данных. Но если один из процессоров записывает в *память*, возникает проблема согласованности кэша: в *кэш*-памяти другого процессора теперь содержатся неверные данные. Необходимо либо обновить *кэш* другого процессора, либо признать содержимое кэша недействительным. Обеспечение однозначности кэшей реализуется на аппаратном уровне – для этого после изменения значения общей переменной все копии этой переменной в кэшах отмечаются как недействительные и последующий *доступ* к переменной потребует обязательного обращения к основной памяти. Необходимость обеспечения когерентности приводит к некоторому снижению скорости вычислений и затрудняет создание систем с достаточно большим количеством процессоров.

Наличие общих данных при выполнении параллельных вычислений приводит к необходимости синхронизации взаимодействия одновременно выполняемых потоков команд. Так, например, если изменение общих данных требует для своего выполнения некоторой последовательности действий, то необходимо обеспечить взаимоисключение с тем, чтобы эти изменения в любой момент времени мог выполнять только один командный *поток*. Задачи взаимоисключения и синхронизации относятся к числу классических проблем, и их рассмотрение при разработке параллельных программ является одним из основных вопросов параллельного программирования.

5. **Согласованность памяти.**

   В системах баз данных согласованность (или корректность) относится к требованию, согласно которому любая данная транзакция с базой данных должна изменять затронутые данные только разрешенными способами. Любые данные, записываемые в базу данных, должны быть действительными в соответствии со всеми определенными правилами, включая ограничения, каскады, триггеры и любую их комбинацию. Это не гарантирует корректность транзакции всеми способами, которые могли потребоваться программисту приложения (за это отвечает код уровня приложения), а просто то, что любые ошибки программирования не могут привести к нарушению каких-либо определенных ограничений базы данных.

Согласованность также можно понимать как то, что после успешной записи, обновления или удаления записи любой запрос на чтение немедленно получает последнее значение записи.

6. **Мультипроцессоры с распределенной памятью. Мультикомпьютер.**

   Во втором типе параллельной архитектуры каждый процессор имеет свою собственную память, доступную только этому процессору. Такая разработка называется мультикомпьютером или системой с распределенной памятью. Она изображена на рис. 8.2, *а*. Мультикомпьютеры обычно (хотя не всегда) являются системами со слабой связью.

   Ключевое отличие мультикомпьютера от мультипроцессора состоит в том, что каждый процессор в мультикомпьютере имеет свою собственную локальную память, к которой этот процессор может обращаться, выполняя команды LOAD и STORE, но никакой другой процессор не может получить доступ к этой памяти с помощью тех же команд LOAD и STORE.

   Таким образом, мультипроцессоры имеют одно физическое адресное пространство, разделяемое всеми процессорами, а муль-тикомпыотеры содержат отдельное физическое адресное пространство для каждого центрального процессора.

   Поскольку процессоры в мультикомпьютере не могут взаимодействовать друг с другом просто путем чтения из общей памяти и записи в общую память, здесь необходим другой механизм взаимодействия. Они посылают друг другу сообщения, используя сеть межсоединений. В качестве примеров мультикомпьютеров можно

назвать IBM SP/2, Intel/Sandia Ootion Red и Wisconsin COW.

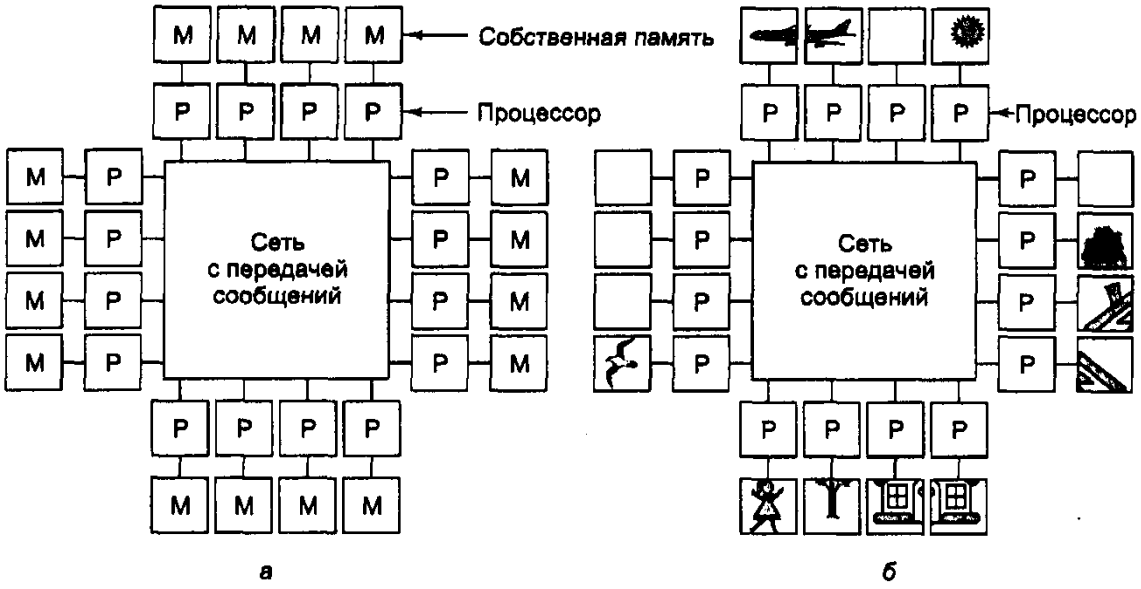


Рис. 8.2. Мультикомпьютер, содержащий 16 процессоров, каждый из которых имеет свою собственную память (*а*); битовое отображение рис. 8.1, разделенное между 16 участками памяти (*б*)

При отсутствии памяти совместного использования в аппаратном обеспечении предполагается определенная структура программного обеспечения. В мультикомпьютере невозможно иметь одно виртуальное адресное пространство, из которого все процессы могут считывать информацию и в которое все процессы могут записывать информацию просто путем выполнения команд LOAD и STORE. Например, если процессор 0 (в верхнем левом углу) на рис. *8Д,б* обнаруживает, что часть его объекта попадает в другую секцию, относящуюся к процессору 1, он может продолжать считывать информацию из памяти, чтобы получить хвост самолета. Однако если процессор 0 на рис. 8.2, обнаруживает это, он не может просто считать информацию из памяти процессора 1. Для получения необходимых данных ему нужно сделать что-то другое.

В частности, ему нужно как-то определить, какой процессор содержит необходимые ему данные, и послать этому процессору сообщение с запросом копии данных. Затем процессор 0 блокируется до получения ответа. Когда процессор 1 получает сообщение, программное обеспечение должно проанализировать его и отправить назад необходимые данные. Когда процессор 0 получает ответное сообщение, программное обеспечение разблокируется и продолжает работу.

В мультикомпьютере для взаимодействия между процессорами часто используются примитивы send и receive. Поэтому программное обеспечение 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. Таким образом, мультикомпьютер программировать гораздо сложнее, чем мультипроцессор.

7. Сетевая система. Сеть рабочих станций.

В компьютерных сетях могут использоваться как однопользовательские мини- и микрокомпьютеры (в том числе и персональные), оснащенные терминальными устройствами для связи с пользователем или выполняющие функции коммутации и маршрутизации сообщений, так и мощные многопользовательские компьютеры (мини-компьютеры, большие компьютеры). Последние выполняют эффективную обработку данных и дистанционно обеспечивают пользователей сети всевозможными информационно-вычислительными ресурсами. В локальных сетях эти функции реализуют серверы и рабочие станции.

Рабочие станции

Рабочая станция (*workstation*) — подключенный к сети компьютер, через который пользователь получает доступ к ее ресурсам. Часто рабочую станцию (равно как и пользователя сети, и даже прикладную задачу, выполняемую в сети) называют клиентом сети. В качестве рабочих станций могут выступать как обычные компьютеры, так и специализированные — «сетевые компьютеры» (NET PC — Network Computer). Рабочая станция сети на базе обычного компьютера функционирует как в сетевом, так и в локальном режимах. Она оснащена собственной операционной системой и обеспечивает пользователя всем необходимым для решения прикладных задач. Рабочие станции иногда специализируют для выполнения графических, инженерных, издательских и других работ. Рабочие станции на базе сетевых компьютеров могут функционировать, как правило, только в сетевом режиме при наличии в сети сервера приложений.. Отличие сетевого компьютера (Network Personal Computer — NET PC) от обычного в том, что он максимально упрощен: классический NET PC не содержит дисковой памяти (часто его называют бездисковым ПК). Он имеет упрощенную материнскую плату, основную память, а из внешних устройств присутствуют только дисплей, клавиатура, мышь и сетевая карта обязательно с чипом ПЗУ BootROM, обеспечивающим возможность удаленной загрузки операционной системы с сервера сети (это классический «тонкий клиент» сети). Для работы, например, в

интранет-сети такой компьютер должен иметь столько вычислительных ресурсов, сколько требует веб-браузер.

Поскольку оставить клиента сети совсем без возможностей локального использования компьютера, например, для работы в текстовом или табличном процессоре со своим персональным «рабочим столом», не совсем гуманно, то иногда используются версии сетевого компьютера, имеющего небольшую дисковую память. Сменные дисководы и флэшдиски должны отсутствовать в целях обеспечения информационной безопасности: чтобы через них не занести в сеть (или вынести) нежелательную информацию — программы, данные, компьютерные вирусы. Конструктивно NET PC выполнены в виде компактного системного блока — подставки под монитор (Network Computer TC фирмы Boundless Technologies) или встроенной в монитор системной платы (NET PC Wintern фирмы Wyse Technology).

8. Три класса приложений. Примеры.

Многопоточные системы

Оконные системы, GUI

-Многопроцессорные операционные системы и системы разделения времени.

-Системы реального времени, управляющие техническими объектами.

Причина — организовать код в виде набора потоков проще, чем в виде большой последовательной программы.

Распределённые вычислени

-Сетевые файловые серверы

-Распределённые системы баз данных

-Web-серверы

-Системы, объединяющие компоненты производства

Отказоустойчивые системы

Причины — интеграция систем, удалённый доступ к данным, повышение надёжности системы.

Синхронные параллельные вычисления

Синхронные параллельные вычисления.

Научные вычисления — математическое моделирование физических процессов в машиностроении, физике, науках о Земле, астрономии, медицине CAD/CAE: FEA, CFD, MBD, optimization

Графика, обработка и синтез изображений

Сложные комбинаторные или оптимизационные задачи, экономическое моделирование.

Причина — ускорение вычислений, решение задач большей размерности.

9. Пять парадигм параллельных программных приложений.

Существует ряд схем взаимодействия самостоятельных частей параллельного приложения. Кратко опишем основные.

Итеративный параллелизм используется для реализации параллелизма в итеративной программе (чаще всего в циклах). Такой параллелизм характерен для распараллеливания по данным в согласованных параллельных вычислениях.

Рекурсивный параллелизм используется в программах с одной или несколькими рекурсивными процедурами, вызов которых независим. Каждый рекурсивный вызов порождает один или несколько новых процессов, которые независимо работают над решением задачи. В рамках такой парадигмы часто реализуются технологии «разделяй и властвуй» или «перебор с возвращением».

Итеративный и рекурсивный параллелизм основан на приемах, известных в последовательном программировании. Следующие схемы взаимодействия характерны именно для параллельных программ.

«*Производители и потребители*» – модель взаимодействия неравноправных процессов по поводу общих данных. Одни процессы «производят» данные, другие – их «потребляют». Часто такие процессы организуются в *конвейер*, через который проходит информация. Каждый процесс конвейера потребляет выход своего предшественника и производит входные данные для своего последователя. Другой распространенный способ организации потоков – древовидная структура, на ней основан, в частности,
принцип *дихотомии*.

«*Клиенты и серверы*» – наиболее распространенная модель взаимодействия процессов в распределенных системах. Клиентский процесс запрашивает (возможно, неоднократно) данные у сервера и ожидает ответа, затем использует полученные данные по своему усмотрению. Серверный процесс ожидает запроса от клиента, далее в соответствии с поступившим запросом обрабатывает данные и возвращает запросившему их процессуклиенту. Таким образом, в отличие от предыдущего случая, между клиентом
И сервером необходимо установить двустороннюю связь. Сервер может быть реализован как одиночный процесс, обслуживающий одновременно несколько клиентских процессов. Сервер может быть многопоточной программой, каждый поток которой обслуживает своего клиента. Если клиент
и сервер выполняются на одном компьютере, то они представляют собой параллельное программное обобщение процедур: сервер исполняет роль процедуры, а клиент ее вызывает. Однако если коды клиента и сервера разнесены в пространстве, то для синхронизации используются

специальные технологии, такие как удаленный вызов процедур или рандеву.

«*Управляющий и рабочие*» – модель организации вычислений, при которой существует поток, координирующий работу всех остальных потоков. Как правило, управляющий поток распределяет данные, собирает и анализирует результаты. Эта парадигма часто применяется в задачах оптимизации и статистической обработки информации, при обработке изображений и других научных вычислениях с итеративными алгоритмами.

«*Взаимодействующие равные*» – модель, в которой исключен не занимающийся непосредственными вычислениями управляющий поток. Распределение работ в таком приложении либо фиксировано заранее, либо динамически определяется во время выполнения. Одним из распространенных способов динамического распределения работ при создании программ для ВС с общей памятью является *портфель задач*. Портфель задач, как правило, реализуется с помощью разделяемой переменной, доступ к которой в один момент времени имеет только один процесс. Если же память ВС распределенная, то такая схема распределения работ превращается в схему «управляющий – рабочий», поскольку портфель задач оформляется отдельным процессом. Основными примерами в этой области являются научные вычисления с итеративными алгоритмами и системы, требующие децентрализованного принятия решений.

10. Базовые языки параллельного программирования. Параллельные операторы, процессы и процедуры.

На чём пишутся параллельные программы.

Уровни

- Программирование GPGPU — CUDA, OpenCL, OpenACC, MS C++ AMP
- API операционных систем — Windows API, POSIX
- RTL -библиотеки языков программирования — C++11, Java, C#
- Дополнительные библиотеки — Intel TBB, MS CCR, Boost, POCO C++ libraries, Qt4

Threads

- Распределённое программирование — MPI, PVM
- RTL -библиотеки с поддержкой компилятора — OpenMP, Cilk Plus
- Фреймворки — Apache Hadoop, BOINC

Инструменты

- MS Visual Studio 2010 Prof, Ulti, 2012 — All (OpenMP)
- Intel Parallel Studio -Intel Parallel Composer, Intel C++ Compiler (OpenMP, Intel Cluster

OpenMP, Cilk Plus)

- GCC — 4.2 (OpenMP)