1. Введение в параллельные вычисления

Мотивы параллелизма

- Параллельность повышает *производительность системы* из-за более эффективного расходования системных ресурсов. Например, во время ожидания появления данных по сети, вычислительная система может использоваться для решения локальных задач.
- Параллельность повышает *отзывчивость приложения*. Если один поток занят расчетом или выполнением каких-то запросов, то другой поток может реагировать на действия пользователя.
- Параллельность *облегчает реализацию* многих приложений. Множество приложений типа «клиент-сервер», «производитель-потребитель» обладают внутренним параллелизмом. Последовательная реализация таких приложений более трудоемка, чем описание функциональности каждого участника по отдельности.

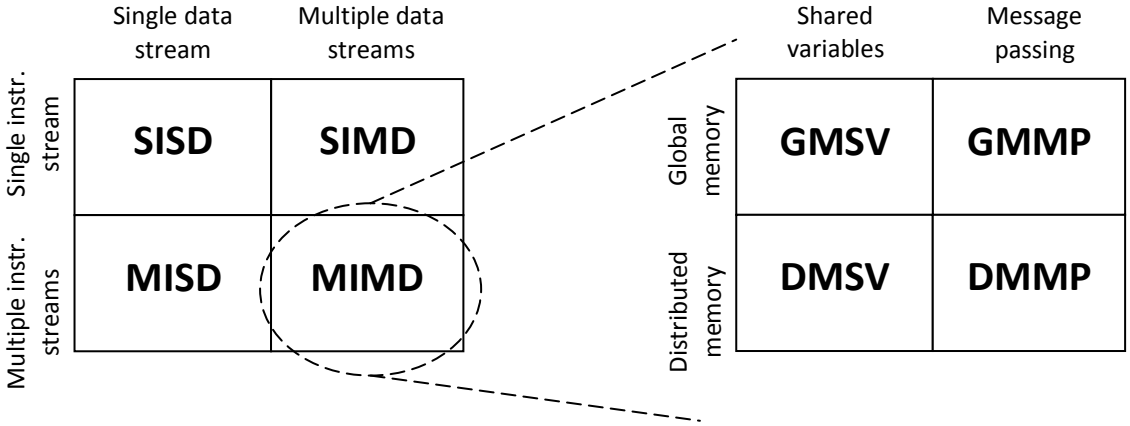
Классификация вычислительных систем

Одной из наиболее распространенных классификаций вычислительных систем является классификация Флинна. Четыре класса вычислительных систем выделяются в соответствие с двумя измерениями – характеристиками систем: поток команд, которые данная архитектура способна выполнить в единицу времени (одиночный или множественный) и поток данных, которые могут быть обработаны в единицу времени (одиночный или множественный).

- **SISD (Single Instruction, Single Data)** – системы, в которых существует одиночный поток команд и одиночный поток данных. В каждый момент времени процессор обрабатывает одиночный поток команд над одиночным потоком данных. К данному типу систем относятся последовательные персональные компьютеры с одноядерными процессорами.
- **SIMD (Single Instruction, Multiple Data)** – системы с одиночным потоком команд и с множественным потоком данных; подобный класс составляют многопроцессорные системы, в которых в каждый момент времени может выполняться одна и та же команда для обработки нескольких информационных элементов. Такая архитектура позволяет выполнять одну арифметическую операцию над элементами вектора. Современные компьютеры реализуют некоторые команды типа SIMD (векторные команды), позволяющие обрабатывать несколько элементов данных за один такт.
- **MISD (Multiple Instructions, Single Data)** – системы, в которых существует множественный поток команд и одиночный поток данных; к данному классу относят систолические вычислительные системы и конвейерные системы;

- **MIMD (Multiple Instructions, Multiple Data)** – системы с множественным потоком команд и множественных потоком данных; к данному классу относится большинство параллельных вычислительных систем.

Классификация Флинна относит почти все параллельные вычислительные системы к одному классу – MIMD. Для выделения разных типов параллельных вычислительных систем применяется классификация Джонсона, в которой дальнейшее разделение многопроцессорных систем основывается на используемых способах организации оперативной памяти в этих системах. Данный подход позволяет различать два важных типа многопроцессорных систем: multiprocessors (мультипроцессорные или системы с общей разделяемой памятью) и multicomputers (мультикомпьютеры или системы с распределенной памятью).



Классификация Флинна Классификация Джонсона

Классификация Джонсоном основана на структуре памяти (global - глобальная или distributed - распределенная) и механизме коммуникаций и синхронизации (shared variables - разделяемые переменные или message passing - передача сообщений). Системы GMSV (global-memory-shared-variables) часто называются также мультипроцессорами с разделяемой памятью (shared-memory multiprocessors). Системы DMMP (distributed-memory-message-passing) также называемые мультикомпьютерами с распределенной памятью (distributed-memory multicomputers).

Архитектура однопроцессорной машины

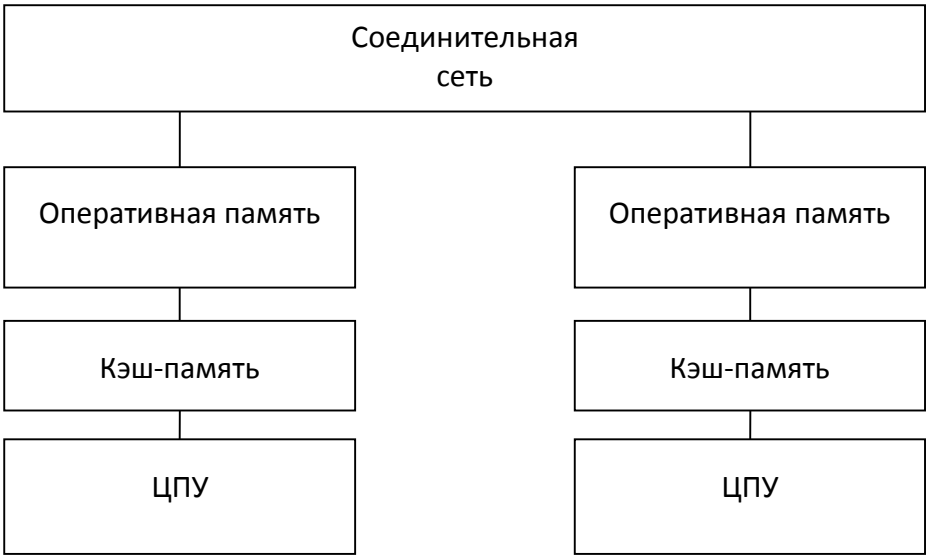
Современная однопроцессорная машина состоит из нескольких компонентов: центрального процессорного устройства (ЦПУ), первичной памяти, одного или нескольких уровней кэш-памяти (кэш), вторичной (дисковой) памяти и набора периферийных

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).

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 процессарами.

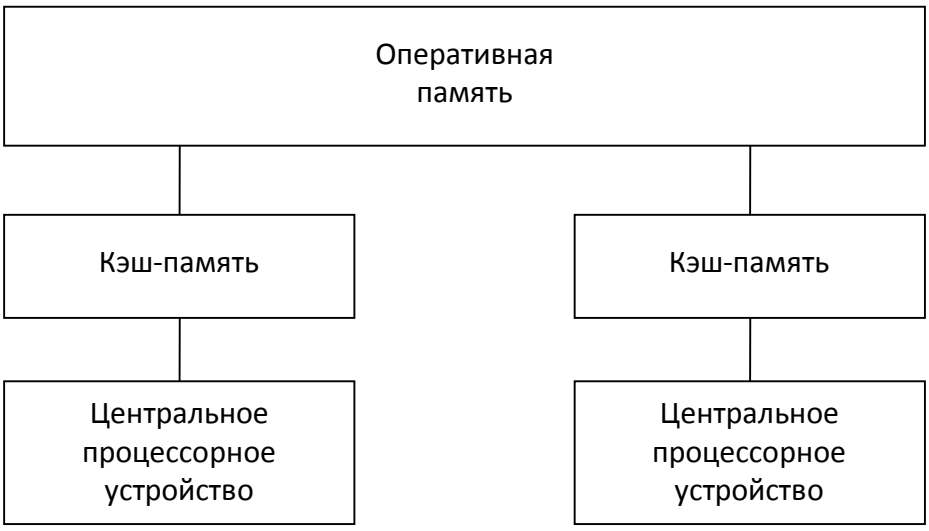
Сетевая система – это многомашинная система с распределенной памятью, связаны с помощью локальной сети или глобальной сети Internet (слабо связанные мультикомпьютеры). Здесь процессоры взаимодействуют также с помощью передачи сообщений, но время их доставки больше, чем в многомашинных системах, и в сети 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.

Режимы выполнения независимых частей программы

При рассмотрении проблемы организации параллельных вычислений следует различать следующие возможные режимы выполнения независимых частей программы:

1) *Режим разделения времени (многозадачный режим)*

Режим разделения времени предполагает, что число подзадач (процессов или потоков одного процесса) больше, чем число исполнительных устройств. Данный режим является псевдопараллельным, когда активным (исполняемым) может быть одна единственная подзадача, а все остальные процессы (потоки) находятся в состоянии ожидания своей очереди на использование процессора; использование режима разделения времени может повысить эффективность организации вычислений (например, если один из процессов не может выполняться из-за ожидания вводимых данных, процессор может быть задействован для выполнения другого, готового к исполнению процесса), кроме того в данном режиме проявляются многие эффекты параллельных вычислений (необходимость взаимоисключения и синхронизации процессов и др.).

Многопоточность приложений в операционных системах с разделением времени применяется даже в однопроцессорных системах. Например, в Windows-приложениях многопоточность повышает отзывчивость приложения – если основной поток занят выполнением каких-то расчетов или запросов, другой поток позволяет реагировать на действия пользователя. Многопоточность упрощает разработку приложения. Каждый поток может планироваться и выполняться независимо. Например, когда пользователь нажимает кнопку мышки персонального компьютера, посылается сигнал процессу, управляющему окном, в котором в данный момент находится курсор мыши. Этот процесс (поток) может выполняться и отвечать на щелчок мыши. Приложения в других окнах могут продолжать при этом свое выполнение в фоновом режиме.

2) *Распределенные вычисления*

Компоненты выполняются на машинах, связанных локальной или глобальной сетью. По этой причине процессы взаимодействуют, обмениваясь сообщениями.

Такие системы пишутся для *распределения обработки* (как в файловых серверах), обеспечения *доступа к удаленным данным* (как в базах данных и в Web), *интеграции и управления данными*, распределенными по своей сути (как в промышленных системах), или *повышения надежности* (как в отказоустойчивых системах). Многие распределенные системы организованы как системы типа клиент-сервер. Например, файловый сервер предоставляет файлы данных для процессов, выполняемых на клиентских машинах. Компоненты распределенных систем часто сами являются многопоточными.

3) *Синхронные параллельные вычисления*.

Их цель – быстро решать данную задачу или за то же время решить большую задачу. Примеры синхронных вычислений:

- научные вычисления, которые моделируют и имитируют такие явления, как глобальный климат, эволюция солнечной системы или результат действия нового лекарства;
- графика и обработка изображений, включая создание спецэффектов в кино;
- крупные комбинаторные или оптимизационные задачи, например, планирование авиаперелетов или экономическое моделирование.

Программы решения таких задач требуют эффективного использования доступных вычислительных ресурсов системы. Число подзадач должно быть оптимизировано с учетом числа исполнительных устройств в системе (процессоров, ядер процессоров).

Уровни параллелизма в многоядерных архитектурах

Параллелизм на уровне команд (Instruction Level Parallelism, ILP) позволяет процессору выполнять несколько команд за один такт. Зависимости между командами ограничивают количество доступных для выполнения команд, снижая объем параллельных вычислений. Технология ILP позволяет процессору переупорядочить команды оптимальным образом с целью исключить остановки вычислительного конвейера и увеличить количество команд, выполняемых процессором за один такт. Современные процессоры поддерживают определенный набор команд, которые могут выполняться параллельно.

Параллелизм на уровне потоков процесса. Потоки позволяют выделить независимые потоки исполнения команд в рамках одного процесса. Потоки поддерживаются на уровне операционной системы. Операционная система распределяет потоки процессов по ядрам процессора с учетом приоритетов. С помощью потоков приложение может максимально задействовать свободные вычислительные ресурсы.

Параллелизм на уровне приложений. Одновременное выполнение нескольких программ осуществляется во всех операционных системах, поддерживающих режим разделения времени. Даже на однопроцессорной системе независимые программы выполняются одновременно. Параллельность достигается за счет выделение каждому приложению кванта процессорного времени.

Анализ эффективности параллельных вычислений

Ускорение параллельного алгоритма по сравнению с последовательным вариантом выполнения определяется как отношение времени выполнения последовательного алгоритма к времени выполнения параллельного алгоритма:

$$S_p = \frac{T_1}{T_p}$$

Эффективность параллельного алгоритма определяется следующим образом:

$$E_p = \frac{T_1}{p \cdot T_p} = \frac{S_p}{p}$$

Эффективность показывает, насколько задействованы вычислительные ресурсы системы; идеальное теоретическое значение эффективности равно единице.

Пределы параллелизма

Достижению максимального ускорения может препятствовать существование в выполняемых вычислениях последовательных расчетов, которые не могут быть распараллелены. Джин Амдал (Gene Amdahl) показал, что верхняя граница для ускорения определяется долей последовательных вычислений алгоритма:

$$S_p \leq \frac{1}{f + (1-f)/p} \leq S^* = \frac{1}{f}$$

f – доля последовательных вычислений в применяемом алгоритме обработки данных, p – число процессоров.

Например, если доля последовательных вычислений составляет 25%, то максимально достижимое ускорение для параллельного алгоритма равно:

$$S^* = \frac{1}{f} = \frac{1}{0.25} = 4$$

В «законе Амдала» имеется несколько допущений, которые в реальных приложениях могут быть неверными. Одно из допущений заключается в том, что доля последовательных расчетов является постоянной величиной и не зависит от вычислительной сложности решаемой задачи. Однако для большинства задач f является убывающей функцией от n, где n – параметр сложности задачи. В этом случае ускорение может быть увеличено при увеличении вычислительной сложности задачи. Нарушение «закона Амдала» также может быть связано с архитектурными особенностями параллельной вычислительной системы. Например, параллельный алгоритм уменьшает объем данных, используемых каждым процессором, и повышает эффективность использования кэш-памяти каждого процессора. Оптимальная работа с кэш-памятью может сильно увеличить быстродействие алгоритма.

Параллельный алгоритм называется масштабируемым, если при росте числа процессоров он обеспечивает увеличение ускорения при сохранении постоянного уровня эффективности использования процессоров.