Литература:

Воеводин В. В. — Параллельные вычисления

Таненбаум — Распределённые вычисления

Под параллельными вычислениями обычно понимают процессы обработки данных, в которых одновременно могут выполняться несколько операций компьютерной системы.

Достижение параллелизма возможно только при соблюдении следующих требований к архитектурным принципам:

1. Независимость функционирования отдельных устройств вычислительной машины, а именно устройств ввода/вывода, процессоров, оперативной памяти.
2. Избыточность элементов вычислительной системы, которая может осуществляться в следующих формах: использование специальных устройств, например, отдельные процессоры для целочисленной и вещественной арифметики; устройства многоуровневой памяти; дублирование устройств, например, использование нескольких однотипных процессоров.

Дополнительной формой обеспечения параллелизма может служить конвейерная реализация обрабатывающих устройств, при которых выполнение операций в устройствах представлено в виде исполнения последовательности, составляющих операцию подкоманд. Таким образом, при вычислениях на таких устройствах на разных стадиях обработки могут находиться одновременно несколько различных элементов данных.

При рассмотрении проблем организации параллельных вычислений необходимо различать следующие возможные режимы выполнения отдельных частей программы:

1. Многозадачные режимы или режимы разделения времени. Он является псевдопараллельным в силу того, что имеется только один процессор, соответственно только один процесс может быть активным.
2. Параллельный режим выполнения. В один и тот же момент времени может выполняться несколько команд обработки данных. Подобный режим обеспечивается, когда есть несколько процессоров или имеются конвейерные и векторные обрабатывающие устройства.
3. Распределённые вычисления. Для параллельной обработки используется несколько устройств, достаточно удалённых друг от друга, в которых передача данных приводит к существенным временным задержкам.

При необходимости использовать распределённые системы важным является разрабатывать такие параллельные алгоритмы, в которых интенсивность потоков межпроцессорных передач данных является низкой.

Проблемы, возникающие при параллельных и распределённых вычислениях

1. Проблемы координации. Если программа содержит подпрограмму или фрагменты, которые могут быть выполнены и эти подпрограммы используют совместно некоторые ресурсы (например, файлы, какие-либо устройства, области оперативной памяти), то это приводит к проблемам координации. Для решения этих проблем необходимо обеспечить связь между задачами и синхронизацию их работы. При некорректной связи или синхронизации обычно возникает четыре проблемы:

   1. Гонка данных (data race). Если несколько задач одновременно пытаются изменить некоторую область данных, а конечное значение зависит от того, какая задача обратиться к этой области первой, то возникает ситуация, которую назвают состоянием гонки (race condition). В том случае, когда несколько задач пытаются обновить один и тот же ресурс данных, такое состояние называют гонкой данных.

   Какая из задач будет выполняться первой, зависит от планировщика задач операционной системы, состояния процессора, времени ожидания и от различных случайных величин. Это и есть состояние гонок.

Для решения подобных проблем применяются определённые правила, например, устанавливается очерёдность для синхронности действий.

2. Бесконечная отсрочка. Это такое планирование, при котором одно или несколько задач должны ожидать до тех пор, пока не произойдёт некоторое событие или не создадутся определённые условия, которые могут оказаться непростыми для реализации. То есть ожидаемые условия или события не являются регулярным или между задачами отсутствует необходимая связь. В результате, если одна или несколько задач ожидают сеанса связи, необходимой для их выполнения, а этот сеанс не происходит, или происходит не полностью, или не происходит вообще, то такие задачи могут не выполниться никогда. Тогда возникает ситуация, называемая бесконечной отсрочкой.

3. Взаимоблокировка (deadlock). Эта проблема также связана с ожиданием и состоит в следующем: если параллельно выполняемые задачи имеют доступ к общим данным, которые им разрешено обновить, то возможна ситуация, когда каждая из задач будет ожидать, пока другая освободит доступ к общим данным. Например: задача В не может освободить данные 1, пока не получит доступ к данным 2, а задача С не может освободить данные 2, пока не получит доступ к данным 1. В результате две задачи оказываются в состоянии взаимной блокировки, могут вести к бесконечной отсрочке. Бесконечная отсрочка и взимоблокировка — самые опасные проблемы.

4. Проблема организации связи. Многие распространённые параллельные среды состоят из гетерогенных компьютерных систем. Гетерогенные компьютерные сети — это системы, которые состоят из компьютеров различных типов, работающие в общем случае под управлением различных ОС, использующих различные протоколы. Процессоры в этих системах могут иметь различную архитектуру, использовать различные машинные языки. Помимо различных ОС эти компьютерные системы могут различаться используемыми стратегиями планирования и системами приоритетов. Более того все эти системы могут различаться параметрами передачи данных. Всё это делает обработку ошибок весьма трудной.

Неоднородные системы зачастую усугубляются и другими различиями, например, может возникнуть необходимость в организации совместного использования данных программами, написанными на различных ЯП и разработанными с использованием различных моделей ПО. Это относится к проблемам межязыковой связи. И даже если параллельная или распределённая среда не является гетерогенной, то всё равно остаются проблемы взаимодействия между процессами или потоками.

В силу того, что каждый процесс имеет собственное адресное пространство, то для совместного использования переменныхи и значений, возвращаемых функциями необходимо применять технологии межпроцессорного взаимодействия. Это образует дополнительный уровень проектирования, тестирования, отладки при создании системы.

Имеется два базовых механизма, которые обеспечивают взаимодействие нескольких задач: общая память и средство передачи сообщений. Для эффективного механизма общей памяти программист должен предусмотреть решения проблем гонки данных, взаимоблокировки, бесконечной отсрочки.

Схема передачи сообщений должна предусматривать возникновение таких неполадок, как прерывистые передачи, потери информации, искажения, слишком длинные сообщения, просроченные и преждевременные сообщения и т.д.

2. Отказы оборудования. Что делать при отказе одного или нескольких процессоров при совместной работе? Должна ли программа завершиться аварийно или перераспределить работу? Что делать при выходе из строя канала связи?
3. Поток данных данных оказывается настолько медленным, что процессы на каждом конце связи превысят выделенный им лимит времени.

Негативные последствия излишнего параллелизма и распределения

Уровень организации может потребовать таких затрат вычислительных ресурсов, что они отрицательно скажутся на производительности задач, совместно решающих одну проблему. В связи с этим возникает следующий

вопрос: как узнать, на сколько процессов, задач или потоков следует разделить программу? Существует ли оптимальное количество процессоров для любой заданной параллельной программы? В какой точке увеличение числа процессоров или компьютеров в системе приведёт к замедлению, а не ускорению работы?

Необходимо различать работу и ресурсы, задействованные в управлении параллельными аппаратными средствами, от работы, направленной на управление потоками.

Предел числа программных процессов может быть достигнут задолго до того, когда будет достигнуто оптимальное число процессоров и компьютеров. Также можно наблюдать снижение эффективности оборудования ещё до достижения оптимального количества параллельно выполняемых задач.

Выбор архитектуры

Архитектурных решений, поддерживающих параллелизм, достаточно много. Архитектурное решение может считаться корректным, если оно соответствует декомпозиции работ ПО. Например, некоторая распределённая архитектура, прекрасно работающая в веб-среде, обречены на неудачу в реальном масштабе времени. В частности, распределённые архитектуры, рассчитанные на длительные временные задержки, приемлемы в веб-среде (e-mail), но не могут применяться, например, в банкоматах.

Выбранная архитектура также может оказывать серьёзное воздействие на параллельную обработку данных, например, методы векторной обработки наилучшим образом подходят для решения определённых математических задач и проблем имитационного моделирования, но совершенно неэффективны в применении к мультиагентным алгоритмам планирования.

Распространённые архитектуры ПО, используемые для поддержки параллельного и распределённого программирования.

Модель	**Архитектура**	**Распределённое программирование**	**Параллельное программирование**
Модель ведущего узла клиент-сервер (главный-подчинённый, управляющий-рабочий)	Есть главный узел, управляющий задачами, который контролирует их выполнение и передаёт работу подчинённым задачам	+	+
Модель равноправных узлов	Все задачи в основном имеют одинаковый ранг и работа между ними распределяется равномерно	-	+
Векторная (конвейерная, поточная) обработка	Один исполнительный узел соответствует каждому элементу массива (вектора) или шагу конвейера	+	+
Дерево с родительскими и дочерними элементами	Динамически генерируемые исполнители типа «родитель-потомок». Данный тип архитектуры полезно использовать в алгоритмых следующих типов: рекурсивные алгоритмы, алгоритмы «разделяй и властвуй», алгоритмы «и-или» и древовидная обработка	+	+

Тестирование и отладка

При параллельном и распределённом программировании трудно воспроизвести точный контекст параллельных и распределённых задач из-за различных стратегий планирования, применяемых в ОС, динамически меняющейся рабочей нагрузки, квантов процессорного времени, приоритетов процессов и потоков, временных задержек при их взаимодействии и собственно выполнении и различных случайных изменений ситуаций, характерных для параллельных или распределённых контекстов.

Чтобы воспроизвести точное состояние, в котором находилась среда при тестировании и отладке, необходимо воссоздать каждую задачу, исполняемую ОС. При этом должен быть известен режим планирования процессорного времени и точно воспроизведены состояния виртуальной памяти и переключения контекстов. Кроме того, следует воссоздать условия возникновения прерываний и формирований сигналов, в некоторых случаях

даже рабочую нагрузку сети. При этом надо понимать, что сами средства тестирования и отладки также оказывают на состояние среды. Всё это означает, что точное воспроизведение событий для тестирования и отладки зачастую невозможно.

Два основных подхода к достижению параллельности вычислений

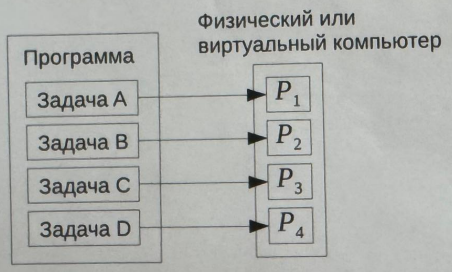
Два базовых подхода к распараллеливанию: параллельное и рапределённое программирование. Эти подходы разные, но могут пересекаться. Параллельное программирование позволяет распределить работу между двумя и более процессорами в рамках одного физического или виртуального компьютера. Распределённое программирование позволяет распределить работу программу между двумя или более процессами (могут быть и на разных компьютерах). OpenMP — многопоточность, MPG — многопроцессорность. Не все распределённые программы включают параллелизм, потому что части программы могут выполняться в различные моменты времени.

Например, программу-календарь можно разделить на две части: одна часть — предоставление информации, присущей календарю, а другая — предоставление сигналов для различных типов встреч. Пользователь составляет расписание встреч, используя одну чать ПО, в то время, как другая часть выполняется независимо от первой. Набор сигналов и компонентов расписания преставляют собой единой приложение, которое разделено на две части, выполняемые по отдельности. При так называемом «чистом» параллелизме одновременно выполняемые части являются компонентами одной и той же программы, а части распределённых приложений обычно реализуются как отдельные программы.

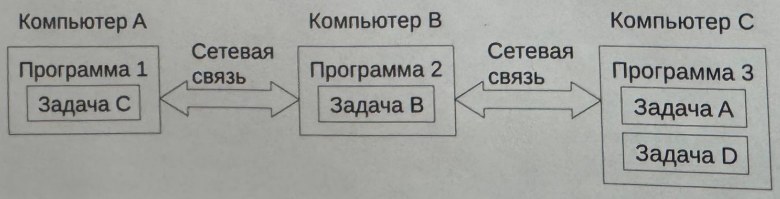
Надо отметить, что не все распределённые программы включают параллелизм, потому что части распределённой программы могут выполняться по различным запросам и в различные периоды времени.

Например, программу Календарь можно разделить на две составляющие: одна часть должна обеспечивать пользователя информацией, присущей календарю, и способом записи данных о важных для него встречах, а другая часть должна предоставлять пользователю набор сигналов для разных типов встреч. Пользователь составляет расписание встреч, используя одну часть программного обеспечения, в то время как другая часть выполняется независимо от первой. Набор сигналов и компонентов расписания представляют собой единое приложение, которое разделено на две части, выполняемые по отдельности. При так называемом чистом параллелизме одновременно выполняемые части являются компонентами одной и той же программы, а части распределённых приложений обычно реализуются как отдельные программы.

а) Параллельное выполнение задачи



б) Распределённое выполнение задачи



Преимущества параллельного программирования