Лекция 1

Современные подходы, классы и особенности

Пути достижения параллелизма:

* Независимость функционирования отдельных устройств компьютеров
* Избыточность элементов вычислительной системы (дублирование, резервирование)
* Конвейерная реализация обрабатывающих устройств

Сдерживающие факторы:

* Высокая стоимость параллельных вычислительных систем
* Необходимость обобщения последовательных программ
* Потери производительности при организации параллелизма
* Постоянное совершенствование последовательных компьютеров
* Зависимость эффективности параллелизма от учета архитектуры параллельных систем

Режимы выполнения независимых частей программы:

* Многозадачный режим (разделение времени). Используется единственные процессор, такой режим псевдопараллельный, так как активный только один процесс, все остальные находятся в очереди и ожидают доступности ресурсов. Использование режима разделения времени позволяет повысить эффективность организации вычислений за счет разделения во времени вычислительных операций и операций связанными с вводом и выводом.
* Параллельные выполнение. Предполагает, что в один момент времени может выполнятся несколько команд обработки данных, такой режим обеспечивается не только при наличии нескольких процессоров, но в случае реализации конвейерных или векторных обрабатывающих устройств.
* Распределенные вычисления. Данный термин используется для указания параллельной обработки данных для которой используется несколько обрабатывающих устройств, удаленных друг от друга, в которых передача данных по линиям связи приводит к существенным временным задержкам. Распределенные вычисления позволяют эффективно обрабатывать данные только для параллельных алгоритмов с низкой интенсивности потоков межпроцессорных передач данных. Такие задачи обычно решаются на многомашинных вычислительных комплексов, когда несколько ЭВМ связываются по локальным или глобальным информационным сетям.

Мультипроцессорные системы (многопроцессорные)

Многопроцессорные системы с общей разделяемой памятью. Такие системы принято называть симметричными многопроцессорными системами. SMP. Основной чертой является то, что каждый процессор имеет прямой и равноправный доступ к любой точке общей памяти. К памяти процессоры подключаются через общую системную шину, при этом упрощается межпроцессорное взаимодействии, упрощается программирование, так как программа работает в едином адресном пространстве.

* + Такая архитектура не пригодна для создания масштабных систем из-за большого числа конфликтов при обращении к общей памяти.
  + Во всех современных машинах иерархическая память. В связи с этим возникает проблема синхронизации информации расположенной в отдельных элементах кэш памяти и основной памяти системы. Современные системы состоят из однородных микропроцессоров с массивом общей памяти. Подключается при помощи шины или комутатора

Для построения масштабируемых систем используются кластерные системы или архитектура типа NUMA. Неоднородный доступ к памяти. В такой архитектуре память распределена физически, но логически общедоступна. С одной стороны это позволяет работать всем процессорам с единым адресным пространством, с другой стороны увеличивает масштабируемость системы, сокращает вероятность конфликтов при доступе к одной и той же ячейке. Когерентность решается аппаратно, в отличии от чистой SMP система с NUM архитектурой вводится 3-х уровневая память:

* Кэш память процессора
* Локальная оперативная память
* Удаленная оперативная память.

Время обращение к элементам памяти разного уровня отличается на порядки, что усложняет написание эффективных параллельных программ. В любой момент времени модули соединенные с помощью высокоскоростного коммутатора могут получать доступ к произвольному элементу удаленной памяти. Масштабируемость NUMa систем ограничивается размером адресного пространства, аппаратуры для поддержки когерентности, и аппаратуры для управления числа процессоров. В настоящее время максимально можно поставить 256 процессоров. Общая производительность около 200 млрд оп\сек.

У любой системы с разделяемой памятью цена растет очень быстро по сравнению с ростом производительности.

Системы с распределенной памятью. MPP

Такие системы представляют собой много процессорные системы с распределенной памяти, которые с помощью коммутационной среды объединяются в выч узлы. Каждый узел состоит из одного или нескольких процессорных элементов, собственной ОЗУ, коммутационного оборудования и подсистемы ввода вывода. Каждый узел обладает всеми компонентами для независимого функционирования. На каждом узле может функционировать либо полноценная ОС, либо ее урезанный вариант, поддерживающие базовые функции ядра.

Процессоры имеют прямой доступ только к своей локальной памяти. Доступ к памяти других узлов реализуется с помощью механизма передачи сообщений. Такая архитектура позволяет устранить конфликты при обращении к памяти и проблему когерентности кэшей. Считается, что имеется возможность наращивания числа процессоров с увеличением производительности.

Основной плюс – высокая масштабируемость.

Система с распределенной памятью подходит для выполнения параллельно независимых программ с малой интенсивностью передачи информации.

Для решения проблем возникающих в выше представленных архитектурах. Решаются векторные системы. В их состав включаются векторные конвейерные системы. Они имеют векторно-конвейерные процессоры. Несколько таких процессоров работают одновременно над общей памятью в рамках многопроцессорных конфигураций. PVP системы являются гибридными системами.

Основной недостаток – цена.

Исследования:

* Исследования в области теории параллельных алгоритмов
* Исследования в области формальных языков методов и технологии параллельного программирования и оптимизации параллельных программ
* Исследования в области теории управления параллельными вычислительными процессами

Эффективность решения конкретной задачи является особенностью реализации параллельности процесса. В целом процесс параллельного решения задачи можно определить как одновременное выполнение отдельных фрагментов задачи, распределенном в вычислительной задаче. Для организации таких вычислений решаемая задача должна обладать свойствами параллелизма.

Свойства параллелизма

* Задачи с естественным параллелизмом. Такие задачи предполагают наличие в процессе своего решения совокупность операции, которые могут выполнятся одновременно без дополнительной передачи информации. Процесс решения задачи с естественным параллелизмом разделяется на несколько процессов, которые выполняются независимости, при этом не требуются специфики вычислений.
* Задачи с параллелизмом множества объектов. Выполняется обработка различных или однотипных объектов по одной и той же программе. В отличии от задач с естественным параллелизмом встречаются в ситуации, когда отдельные участки вычислений должны выделятся по разному для объектов. Основной характеристикой является ранг т.е. количество параметров по которым должна вестись обработка. В качестве ранга задачи может выступать количество разнотипных объектов в системе.
* Задачи с параллелизмом независимых ветвей. Наиболее разработанные тип параллелизма. При решении крупных задач могут быть выделены независимые части или ветви. Условия независимости ветви задачи У от ветки Х являются:
  + Между ветвями отсутствуют функциональные связи. Выход Х не является вход Y
  + Между ветвями нет связи по памяти
  + Независимость в программном отношении
  + Взаимонезависимость по управлению.

Все задачи решаемые одновременно могут быть связаны по данным и по управлению в общей вычислительной среде. При этом задачи разделяются по степени связывания. Степень связывания определяет зернистость параллелизма. Пусть существует некоторая сложная задача, которая может быть разбита но множество простых задач, тогда задача называется несвязанной если все элементы матрицы связности являются нулевыми. Одиночные задачи соединяются в сложную несвязную задачу в виде набора простых, вычисления при решении простых задач ведутся независимо друг от друга, обмен происходит довольно редко. Объектом распралалеливания являются простые задачи. Такие вычисления являются крупно зернистыми.

Если часть элементов матрицы связности равны нулю, то сложная задача состоящая из набора простых называется слабо связанной. Такие задачи образуют обширный класс, в котором простые задачи объединяются либо функциональными связями, либо связями по данным. Принято считать, что общий объем взаимодействия по данным меньше объема вычисления, то появляющийся набор простых задач можно считать общей слабосвязной задачей. Наличие связности между задачами не позволяет эффективно решать ее на распределенных устройствах. Объектом распараллеливания является набор вызываемых процедур. Организация управления вычисления определяется программистом. Такую организацию принято называть средне-зернистым параллелизмом.

Сложная задача, состоящая из набора простых называют сильно связанное, если все элементы имеют не нулевые значения. В сильносвязных задачах число обменных взаимодействий по управлению данных сопоставимо с объемом вычислением в простых задачах. В предельном случае на каждом шаге вычислений выполняется обмен данными между простыми задачами. Распараллеливание выполняется на уровне внутренних блоков (циклов). Это мелкозернистая организация.

Методы и средства организации параллельных вычислений определяются уровнем организации параллельных вычислений. Традиционно принято выделять 4 уровня:

* Уровень заданий.
* Уровень программ.
* Уровень команд. Реализуется средствами аппаратного ядра ОС.
* Арифметический уровень. Обеспечивается параллельное или конвейерное исполнение команды. Реализуется аппаратными средствами.

Модели параллельного программирования

Определяют набор свойств, присущие разрабатываемому алгоритму для создания максимально эффективной программы. Основными свойствами являются:

* Параллелизм. Возможность выполнят несколько действий одновременно
* Масштабируемость. Возможность увеличения производительности при увеличении числа процессоров
* Локальность. Предпочтения использование локальной памяти по сравнению с удаленной
* Модульность. Декомпозиция сложных программ на простые компоненты

Модели:

* Задача\канал
* Передача сообщения
* Разделяемая общая память
* Параллелизм данных

В зависимости от свойства задачи, уровня зернистости и способа организации параллельных вычислений используется ряд программных моделей. При этом под программной моделью принимается совокупность технологий и приемов программирования, структур обрабатываемых и генерируемых в процессе вычисления данных.

Информационный граф

При выборе программной модели необходимо предварительно проанализировать разрабатываемый алгоритм. Для анализа и эффективности алгоритма наиболее часто применяют язык П-схем, параллельные формы и информационные графы. Используют графы операции-операнд, так и графы процесс-ресурс.

Информационный граф используется для определения возможности расрпеделения и времени выполнения алгоритма заданной задачи. Он описывает последовательность выполнения операции и зависимость между отдельными операциями. Узлами информационного графа могут являться как отдельные операции, так и их совокупность. Однонаправленные дуги определяют каналы обмена данными между узлами. При такой интерпретации в каждом листе графа может быть оператор. Каждый узел задается парой n и s, n – имя, s – размер. Размер определяет количество простейших операций, входящих в состав этого узла. Дуга так же характеризуется этими значениями (v, d). v – пересылаемая переменная, d – время передачи. В процессе разработки алгоритма может выполняться слияние нескольких узлов, такой операцией называют упаковкой графа. Она с одной стороны уменьшает детализацию алгоритма, но увеличивает количество дуг, входящих и выходящих из вершины. Упаковка позволяет скрыть несущественные на данном этапе разработки коммутационные затраты. Тем самым снизить затраты на вычисления.

Ярусно-параллельная форма.

Предложена Поспеловым. Согласно ЯПФ любой алгоритм или выч процесс можно представить в виде набора ярусов, причем каждый следующий ярус входят только зависимые ветви предыдущих ярусов. На каждом ярусе операции могут выполнятся параллельно и независимо друг от друга задачи. При этом сплошными стрелками определяются функциональные зависимости, связанные с передачей данных, а пунктирными стрелками указаны управляющие связи. Для оценки сложности и эффективности параллельных вычислений, вводятся 2 понятия:

* Ширина i-го яруса.
* Ширина графа ЯПФ. Макмимальная ширина входящих ярусов.

При различных способах организации вычислений эти показатели могут варьироваться. Тогда вводят дополнительный показатель – ширина i-го яруса при j-реализации. Основной недостаток является, то что они относятся не к задаче, а к схеме ее решения. Для одной и той же задачи можно получить разные характеристики.

Традиционно при проектировании алгоритмов используется последовательный способ. В любой момент времени в системе выполняется только одна операции, только на определенных наборах данных.

Параллельная модель программирования обеспечивает возможность учета особенностей вычислительного процесса в среде с множеством одновременно работающих вычислителей. При этом должны быть решены задачи организации обмена данными, размещение данных в общей или разделяемой памяти. С программной точки зрения параллельные программы для организации параллельных вычислений должны обладать 4 свойствами:

* Параллелизм. Способность выполнять несколько действий одновременно
* Масштабируемость. Возможность увеличения производительности, при увеличении числа процессоров.
* Локальность. Предпочтительность в использовании локальной памяти, к удаленной
* Модульность. Возможность декомпозиции сложных программ на простые компоненты.

Получаемое при этом время решение задачи должно быть оптимальным и приемлимым для заказчика

Модели параллельного программирования

Существует всего 4 модели:

* Модель задача\канал. Простейшая модель параллельного программирования, при которой программа состоит из нескольких простых задач. Задачи связанные между собой коммутационными каналами и могут выполняться одновременно. Каждая простейшая задача состоит из последовательного кода и локальной машины. Таким образом задача использует собственную локальную память и собственный набор входных и выходных портов. Набор портов определяет интерфейс задач с ее окружением. При такой постановке каждый коммутационный канал интерпретируется как очередь сообщений. Каждая задача имеет собственную очередь и способна поместить туда сообщение и удалить. В процессе решения общей задачи, количество подзадач может меняться. Что приводит у изменению количества коммутационных каналов. В данной модели операции отправки и принятия сообщений работают по разному. В частности операция отправки – асинхронная, а приема сообщений – синхронная. При разработке алгоритмов согласно модели канал\задачи нет необходимости учитывать количество процессоров, закладка происходит по максимуму, что может привести к снижению производительности.
* Модель передачи сообщений. Согласно данной модели проектируются вычислительные процессы, которые могут порождать несколько других подпроцессов, при этом каждый простой задаче присваивается индетификатор. Взаимодействие между задачами происходит при передачи путем точка-точка. Сообщение посылается не в канал, а подзадаче. Модель передачи сообщения предполагает динамическое создание подзадач, выполнение нескольлких подзадач на процессоре. Simple program multiple data – SPMD. Каждая задача содержит один код но разные данные. Согласно данной модели работают библиотеки технологии MPI. Создается фиксированное число задач, не поддерживается динамическое управление задачи, необходимо наличие множественных данных.
* Модель разделяемой общей памяти. Основана на использовании одной операции приминительно к множеству элементов обрабатываемой структуры. Задача программирования в рамке данной модели содержит последовательность операции над каждым элементом данных. При этом каждая операция должна быть независима. Используется в случае если удается раздели адресное пространство, чтение и запись в которые осуществляется асинхронно. Во избежание конфликтов используются механизмы блокировок и семафорные операции. Данные являются общедоступными, что упрощает разработку программы. Локализация данных не привествуется.
* Модель параллельной обработки данных. Предполагает чистую векторизацию, используется для выполнения операций над массивами данных.

Проектирование любого параллельного программирования выполняется в 4 этапа:

* Декомпозиция
* Планирование коммутаций
* Укрупнение
* Планирование вычислений

Декомпозиция

На данном этапе выполняется анализ оценки возможности распараллеливания. Задачи и связанные с ней данные разбиваются на более мелкие подзадачи. Данное действие выполняется без учета архитектуры

Требования:

* Количество задач должно быть большим.
* Избегать количество вычислений и пересылок данных
* Подзадачи должны быть примерно одинакогого рахмера

Выделяют 2 типа:

* Декомпозиция по данным. Предполагает первоначальное сегментирование данных. Т.е. выделение элементерных частиц подлежащих параллельной обработки.
* Фукнциональная декомпозиция. Изначально определяются операции, которые могут выполняться независимо друг от друга. А затем определяется объем передаваемой информации между опеарциями.

Типы декомпозиции вычисляются вычислительными схемами:

* Параллелизм данных. Выполняется параллельная обработка больших объемов данных. Используется естественный параллелизм задачи. В этом случае если параллельной обработкой управляет только одна программа, имеет место рад особенностей организации выч процесса:
  + Пространство имен должно являться глобальным. Т.е. детали структуры данных, методов организации доступа к памяти и протоколы межпроцессорного обмена должны быть скрыты.
  + Применяется слабая синхронизация на параллельно работающих процессорах. Выполнение команд осуществляется независимо.
  + Отдельный процессор выполняет один и тот же фрагмент программы, но отсуствует гарантии того, что в заданный момент времени на всех процессорах выполняется одна и та же операция.
  + Программа с паралеллизмом данных состоит из последовательности операции.

При применении паралелизма данных используются специальные языки C\*, Fortran. Реализация поддержки паралеллизма данных поддерживается на уровне транслятора

* + Пропроцессоры. Специальные библиотеки, которые поддерживают распараллеливание
  + Предтрансляторы. Выполняют предварительный анализ логической структуры программы, проверку зависимости программных операций и ограничений на параллелизм.
  + Распаралелливающие трансляторы. Выявляют параллелизм в исходном коде программы и преобразуют в паралелльные конструкции. При этом в программу добавляется специальная директива.

Программист:

* + Задание директив
  + Задание опций оптимизаций
* Функциональный параллелизм (задач). Выполняются разные задачи над одним и тем же набором данных. Подход подразумевает, что вычислительная задача разбивается на самостоятельные задачи. Параллелизм задач ориентирован на .. Чем большие число подзадач, тем больше число процессоров можно использовать. При этом для каждой подзадачи пишется решение на ЯП. Подзадачи должны обменивать результатами.

Для построение эффективного кода необходимо проводить анализ затрат, с целью выявления ресурсоподребляющих частей.

Данная модель более трудоемкая, отвественность ложится на программиста, он обязан минимизировать обмен данных между задачами и избегать тупиковых случаев. Достоинства:

* + Гибкость, максимальная эффективность.
  + Максимальное быстродействие.
* Формирование максимального возможного количества подзадач:
  + Предельно допустымый параллелизм
  + Затруднение анализа параллельных вычислений
* Использование крупных подзача
  + Ясная схема
  + Затруднение эффективного использование большого числа процессоров
* Промежуточный уровень.

Независимость:

* По данным. Данные обрабатываемые одной частью программы не модифицируются другой частью.
* По управлению. Порядок выполнения частей программы может быть определен только во время выполнения программы, зависимые программы всегда последовательными.
* По ресурсам. Обеспечивается достаточным количеством вычислительных ресурсов.
* По выводу. Возникает, если 2 подзадачи не производит запись в одну и ту же переменную.

В целом можно считать, что декомпозиция выполнена удачно, если время вычисления или обработки данных превосходит время необходимое для пересылки данных.

Проектирование коммутаций

На данном этапе определяется все необходимые взаимодействия между подзадачами. Проектирование коммутации заключается в определении структуры коммутационных каналов, которые напрямую или косвенно связывают поставщика информации с потребителем этой информации. Сами каналы могут создаваться в явном или неявном виде.

Сети коммутации принято подразделять на следующие типы:

* Локальные и глобальные.
  + Для локальных в каждый момент времени передача выполняется между небольшим набором подзадач
  + Для глобальных при передаче информации участвует большое число подзадач, задействована максимальная коммутация.
* Структурные схемы и схемы с произвольным способом взаимодействия.
  + В случае структурной. Каждая задача и подзадача образует регулярную структуру с заданной топологией.
  + Для произвольных передача данных неоднородна, топологии связанные с подзадаче представляются в виде графа.
* Статические и динамические схемы.
  + Статические источник и получаетель, а так же время фиксируются на этапе проектирования
  + В динамике все выполняется в динамике
* Синхронные и асинхронные
  + Для синхронных все ждут
  + Для асинхронного возможен вариант без ожидания

Рекомендации по проектированию каналов:

* Количество коммуникаций у подзадач должно быть примерно одинаковым
* Максимально возможно использование локальных коммутаций
* Отдавать предпочтения структурных коммутациям, так как они могут организовываться параллельно
* Обмен данными между позадачами должен учитывать архитектуру среды, обеспечивая равномерную загрузку и минимизируя конфликты.

Одним из наиболее распространенных способов – использование библиотек реализующие обмен сообщениями. Данные библиотеки позволяют реализовать коммутации для разных платформ.

Другим подходом является использование технологии rpc.

Укрупнение

Данный этап учитывае архитектуру вычислительно устроства, на котором будет выполнятся задача. Масштабирование проводится в случае, если количество имеющихся подзадач не совпадает с чиисло доступных процессоров.

Для сокращения количества подзадач необходимо выполнить укрупнению

* Уменьшение размерности декомпозиции
* Объединение смежных задач – повышение степени детализации
* Объединение поддеревьев в структуре «разделяй и властвуй»
* Объединение узлов в дереве. Сокращает количество используемых процессорных переменных.

Укрупнение приводит к увеличению зернистости и иногда к снижению стоимости подпрограмм.

* Снизить затраты на коммутацию.
* Не допустить снижение скорости вычислений и масштабируемости программ.
* Выравнивание трудоемкости подзадач.
* Снижение стоимости и трудоемкости разработки.

Планирование вычислений

Заключительный этап планирования алгоритмов. Производит распределение задач между процессами. Основной критерий – эффективность использование процессорных элементов с минимизации затрат на обмен информаций.

Определяется на каком из процессоров будет выполняться операция и в какой последовательности. Максимально просто решается при использований систем разделяемой памяти. В алгоритмах основанных на функциональной декомпозиции часто реализуются коротко живущие подзадачи. В этом случае применяются алгоритмы планирования подзадачи, которые распределяют задачи по незагруженным процессорам. Порядок доступа должен быть определен на уровне алгоритма.

Схемы планирования вычислений:

* Хозяин-работник. Согласно этой схеме все вычислительные подзадачи подразделяются на работников, которые выполняют набор действий, результат которых возвращается главной задаче.
* Иерархическая схема. Главная задача, которая запускает подчиненные задачи, которые запускают подчиненные задачи, которые запускают подчиненые задачи, которые запускают подчиненные задачи, которые запускают подчиненные задачи, которые запускают подчиненные задачи.
* Децентрализованная схема. В данной схеме нет главной задачи, они могут обмениваться данными, придерживаясь определенной стратегии.

При решении сложных задач применяются динамические методы планирования. Эти методы эффективно показывают себя в следующих случаях:

* Если каждый процессор выполняет только одну поздалачу, тогда длительность выполнения подзадачи расчитывается по самой длиной
* Оптимальная производительность достигается только в том случае, если все задачи имеют одинаковую трудоемкость
* Сбалансированность может быть обеспечена посредством догрузки отдельным процессоров дополнительными подзадачами.

Существует ряд методов динамической сбалансированной загрузки:

* Метод рекурсивной кординатной дихотомии. Данный метод применяется к задачам на нерегулярной сетке для которых применяется параллелизм данных. Разрез сетки на каждом шаге выполняется вдоль того, где сетка имеет максимальную протяженность. Данный метод не позволяет добиться высокой коммутации.
* Метод рекурсивной дихотомии графов. Так же используют для регулярных сеток. Минимизация ребер приводит к сокращению числа коммутаций.

Алгоритмы используются для планирования вычислений с применением глобальной коммутации. Для локальной коммутации применяют вероятностные методы планирования. При вероятностном – задача размещается случайным образом, что если число задач велико, то средняя нагрузка на процессор приблизительно одинакова.

При организации параллельных вычислений возникают 4 проблеммы:

* Синхронизация. Когда несколько модулей одной и той же программы запускаются на разных процессорах. Необходимо синхронизация их действий. Она необходима как для доступа к памяти, так и для доступа к ресурсам вычислительной системы. Синхронизация определяет какой из модулей может первым обратиться к ресурсам.
* Проблема гонка данных. Возникает в вычислительной системе с общей памятью, когда процессоры работают параллельно в и одно и тоже время должны получить доступ к одним и тем же данным. Совместная запись недопустима. Чтобы справиться с проблемой гонок используется механизм обращения к памяти. Используются семафоры и тд.
* Проблема deadlock. Чтобы исключить проблему гонки включают блокировку. Блокировка является опасным механизмом. Блокировка может возникнуть в случае одновременной остановки взаимозависимых задач. Механизмы обхода тупиков. Включение дополнительных операций, позволяющих снять ранее возникшие блокировки, либо увеличивающих объем кода с целью снятия необходимости одновременного захватывания.
* Проблема послать-получить. Возникает чаще всего в мультикомпьютерных комплексах, где каждый компьютер имеет собственную память. Во время решения общей задачи требуется обмен сообщений. Передача данных является наиболее медленной операцией, поэтому при анализе эффективности работы программы важную роль играют посылки и получение сообщений. Решением является блокирующие и неблокирующие примитивы.