Щедрин Георгий Александрович, Мардоян Михаил Максимович

ОРГАНИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ДЛЯ МОДЕЛИРОВАНИЯ ЗАМКНУТЫХ СИСТЕМ УПРАВЛЕНИЯ

В статье рассматриваются способы и алгоритмы моделирования систем автоматического управления с помощью параллельных систем обработки информации. Представлены ряд оптимизаций и моделей алгоритмов организации параллельных вычислений.

Адрес статьи: www.gramota.net/materials/1/2012/6/60.html

Источник

Альманах современной науки и образования

Тамбов: Грамота, 2012. № 6 (61). С. 182-186. ISSN 1993-5552.

Адрес журнала: www.gramota.net/editions/1.html

Содержание данного номера журнала: www.gramota.net/materials/1/2012/6/

15. Gothic FAQ by Andrew «Coroner» Narkevich [Электронный ресурс]. URL: http://old.gothic.ru/music/gothfaq.htm (дата обращения: 18.04.2012).
16. Hodkinson P. Goth: Identity, Style and Subculture. Oxford, 2002. 219 p.
17. Hogle J. E. Cambridge Companion to Gothic Fiction. New York, 2002. 327 p.
18. http://www.buzzle.com/articles/dark-romanticism.html
19. http://www.gothicsubculture.com/
20. Issitt M. L. Goths: a Guide to an American Subculture. California, 2011. 159 p.
21. Jasper A. «I am Not a Goth!». The Unspoken Morale of Authenticity within the Dutch Gothic Subculture // ETNOFOOR. 2004. XVII (1/2). P. 90-115.
22. Jasper A. Mysts of the Gothic Underground: an Endo-Ethnographic Perspective on Gothic Subculture: Identity, Consumerism and the Gothic Aesthetic from the Inside Out. University of Amsterdam, 2006. 157 p.
23. Kilpatrick N. The Goth Bible: a Compendium for the Darkly Inclined. St. Martin's Griffin, 2004. 304 p.
24. Koster D. N. Influences of Transcendentalism on American Life and Literature // Literary Movements for Students. Detroit: Thompson Gale, 2002. Vol. 1. 336 p.
25. Punter D., Byron G. The Gothic. Oxford, UK, 2004. 315 p.
26. Spooner C. Contemporary Gothic. L., 2006. 175 p.
27. Spooner C., McEvoy E. Routledge Companion to Gothic. New York, 2007. 290 p.
28. Thomas S. A. Subjectivity in American Popular Metal: Contemporary Gothic, the Body, the Grotesque, and the Child. University of Glasgow, 2009. 219 p.
29. Thompson G. R. Introduction: Romanticism and the Gothic Tradition // Gothic Imagination: Essays in Dark Romanticism. Pullman, WA: Washington State University Press, 1974.

УДК 681.5
Технические науки

В статье рассматриваются способы и алгоритмы моделирования систем автоматического управления с помощью параллельных систем обработки информации. Представлены ряд оптимизаций и моделей алгоритмов организации параллельных вычислений.

Ключевые слова и фразы: особенности организации параллельных вычислений; алгоритмы; методы; среда параллельной обработки информации; *PPL*.

Георгий Александрович Щедрин
Кафедра информатики и информационных технологий
Пятигорский государственный гуманитарно-технологический университет
schedrin@smtp.ru

Михаил Максимович Мардоян
Центр информационных технологий
Пятигорский государственный гуманитарно-технологический университет
schedrin@smtp.ru

ОРГАНИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ДЛЯ МОДЕЛИРОВАНИЯ ЗАМКНУТЫХ СИСТЕМ УПРАВЛЕНИЯ©

В наше время персональные компьютеры получили широкое применение в различных сферах деятельности человека. Достижениями в отрасли информационных технологий сегодня вряд ли кого-то удивишь, настолько мы все привыкли к ним в повседневной деятельности. Излишне говорить, что информационные технологии сегодня играют слишком важную роль в нашей жизни, чтобы попытаться их игнорировать. За примером далеко ходить не надо - попробуйте сегодня найти область человеческой деятельности, так или иначе, не связанную с компьютером. Если рассматривать область математического моделирования замкнутых систем управления, то стоит обратить внимание на чрезмерно математически трудный аппарат. Например, при моделировании системы управления на основе функции Грина происходит сотни операций в секунду [1-4]. Однако если вести расчет для многомерных распределенных систем, то возникает сложность математического моделирования таких процедур. Выходом из сложившейся ситуации, на наш взгляд является применение параллельных алгоритмов и структур для реализации данной задачи. Рассмотрим основные принципы построения и историю возникновения параллельных алгоритмов на основе на основе некоторых языков программирования.

Параллельное программирование возникло в 1962 году с изобретением каналов - независимых аппаратных контролеров, позволявших центральному процессору выполнять новую прикладную программу одновременно с программами ввода-вывода других (приостановленных) программ. Параллельное программирование (слово

параллельное в данном случае означает «происходящее одновременно») первоначально было уделом разработчиков операционных систем. В конце 60-х годов были созданы многопроцессорные машины. В результате были поставлены не только новые задачи разработчикам операционных систем, но и появились новые возможности у прикладных программистов. Первой важной задачей параллельного программирования стало решение проблемы, так называемой критической секции. Эта и сопутствующие ей задачи («обедающих философов», «читателей и писателей» и т.д.) привели к появлению в 60-е годы огромного числа научных работ. Для решения данной проблемы и упрощения работы программиста были разработаны такие элементы синхронизации, как семафоры и мониторы. К середине 70-х годов стало ясно, что для преодоления сложности, присущей параллельным программам, необходимо использовать формальные методы. На рубеже 70-х и 80-х годов появились компьютерные сети. Сети привели к росту распределенного программирования, которое стало основной темой в 80-х, и особенно, в 90-х годах. Суть распределенного программирования состоит во взаимодействии процессов путем передачи сообщений, а не записи и чтения разделяемых переменных.

Сейчас, на заре нового века, стала заметной необходимость обработки с массовой организацией параллельных вычислений, при которой для решения одной задачи используются десятки, сотни и даже тысячи процессоров. Также видна потребность в технологии клиент-сервер, сети *Internet* и *World Wide Web*. Наконец, стали появляться многопроцессорные рабочие станции и ПК. Параллельное аппаратное обеспечение используется больше, чем когда-либо, а параллельное программирование становится необходимым. Идеи параллельных и распределенных вычислений сегодня распространены повсеместно. Как обычно в вычислительной технике, прогресс исходит от разработчиков аппаратного обеспечения, которые создают все более быстрые, большие и мощные компьютеры и коммуникационные сети. Большая часть мира вычислительной техники сейчас параллельна. Параллельное программирование возникло в 60-х годах в сфере операционных систем. Причиной стало изобретение аппаратных модулей, названных каналами, или контроллерами устройств. Они работают независимо от управляющего процессора и позволяют выполнять операции ввода-вывода параллельно с инструкциями центрального процессора. Канал взаимодействует с процессором с помощью прерывания - аппаратного сигнала, который говорит: «останови свою работу и начни выполнять другую последовательность инструкций».

Результатом появления каналов стала проблема программирования (настоящая интеллектуальная проблема) - теперь части программы могли быть выполнены в непредсказуемом порядке. Следовательно, пока одна часть программы обновляет значение некоторой переменной, может возникнуть прерывание, приводящее к выполнению другой части программы, которая тоже пытается изменить значение этой переменной.

Вскоре после изобретения каналов началась разработка многопроцессорных машин, хотя в течение двух десятилетий они были слишком дороги для широкого использования. Однако сейчас все крупные машины являются многопроцессорными, а самые большие имеют сотни процессоров и часто называются машинами с массовым организацией параллельных вычислений (massively parallel processor). Скоро даже персональные компьютеры будут иметь несколько процессоров. Многопроцессорные машины позволяют разным прикладным программам выполняться одновременно на разных процессорах. Они также ускоряют выполнение приложения, если оно написано (или переписано) для многопроцессорной машины. При использовании каналов и многопроцессорных систем возникают и возможности, и трудности. При написании параллельной программы необходимо решать, сколько процессов и какого типа нужно использовать, и как они должны взаимодействовать. Эти решения зависят как от конкретного приложения, так и от аппаратного обеспечения, на котором будет выполняться программа. В любом случае ключом к созданию корректной программы является правильная синхронизация взаимодействия процессов.

Особенности организации параллельных вычислений

При параллельных вычислениях необходимо программировать специальные действия по координации работы задач, а также следует четко определить «область деятельности» для каждой задачи [5]. Рассмотрим здесь несколько возможных подходов для решения этих проблем.

При первом варианте организации параллельных вычислений все задачи запускаются одной командой, в которой указывается имя запускаемого исполняемого файла, число запускаемых задач, а также число и тип используемых процессоров. По этой команде запускаются на указанных процессорах требуемое число копий указанного исполняемого файла. Следовательно, программные коды всех запущенных в *PVM* задач в данном случае одинаковы. Для того чтобы эти задачи могли выполнять разные действия, им должен быть известен признак, отличающий каждую задачу от остальных. Тогда, используя этот признак в условных операторах, легко запрограммировать выполнение задачами разных действий. Наличие такого признака предусмотрено в любой многозадачной операционной системе. Это так называемый идентификатор задачи - целое число, которое присваивается задаче при запуске. Существенно, что при запуске задача получает идентификатор, отличный от идентификаторов задач, выполняемых в данное время. Это гарантирует, что идентификаторы всех запущенных задач в *PVM* будут различными. Если теперь обеспечить задачи возможностью определять собственный идентификатор и обмениваться информацией с другими задачами, то ясно, что им легко распределить между собой вычислительную работу, в зависимости, например, от занимаемого места в упорядоченном наборе идентификаторов задач.

Во втором обычно используемом варианте запуска задач сначала запускается одна задача (master), которая в коллективе задач будет играть функции координатора работ. Эта задача производит некоторые подготовительные действия, после чего запускает остальные задачи (slaves), которым может соответствовать либо

тот же исполняемый файл, либо разные исполняемые файлы. Такой вариант организации параллельных вычислений предпочтительнее при усложнении логики управления вычислительным процессом, а также когда алгоритмы, реализованные в разных задачах, существенно различаются или имеется большой объем операций (например, ввода - вывода), которые обслуживают вычислительный процесс в целом.

Оптимизация параллельных вычислений

Главным критерием качества распараллеливания вычислений является сокращение общего времени решения задачи. Для простоты в качестве задач будем рассматривать здесь вычисления по формулам, в которых все алгебраические операции выполняются за одинаковое время T, а используемые процессоры все имеют одинаковые характеристики.

Пусть требуется вычислить величины

Y1 = A * B + C * D

Y2 = (A + B) * C + D

При использовании одного процессора Y1 и Y2 будут вычислены за одинаковое время 3T. При использовании двух процессоров Y1 может быть вычислено за время 2T, т.к. операции умножения могут быть выполнены параллельно. А вычисление Y2 не может быть ускорено, т.к. сама формула не допускает параллельного выполнения операций.

Из рассмотренного примера следует, что возможности для распараллеливания вычислений ограничиваются не только числом имеющихся процессоров, но и особенностями вычислительного алгоритма, который может вообще не допускать распараллеливания. Кроме того, ясно, что предельный эффект от использования N процессоров может быть достигнут лишь тогда, когда общая вычислительная нагрузка может быть разделена точно на N частей, и при этом ни один из процессоров не простаивает от начала и до конца вычислений.

Теперь обратим внимание на то, что в рассмотренном примере были проигнорированы затраты времени на пересылки данных между процессорами. Это отчасти правомерно, если процессоры используют общую память (следует отметить, что все процессоры имеют индивидуальную регистровую память гораздо более быстродействующую, чем ОЗУ). Если же процессоры работают с данными, хранящимися в индивидуальных ОЗУ, то обмен данными между этими ОЗУ снижает эффект от распараллеливания вычислений вплоть до того, что этот эффект может стать отрицательным. Отсюда следует, что при оптимизации распараллеливания вычислений нужно учитывать время, затрачиваемое на обмен данными между процессорами. И вообще при прочих равных условиях уменьшение числа и объемов сообщений, которыми обмениваются параллельно работающие процессоры, как правило, приводит к сокращению общего времени решения задачи. Поэтому желательно распараллеливать вычислительный процесс на уровне как можно более крупных блоков алгоритма.

До сих пор мы рассматривали вычисления, производимые над одной порцией исходных данных. При этом распараллеливание базируется на декомпозиции алгоритма вычислений. Более богатые возможности для распараллеливания открываются при многократном использовании формулы для потока входных данных. При этом, как правило, удается достичь максимального эффекта от распараллеливания вычислений, что видно на примере вычислений по формулам для Y1 и Y2.

Пусть каждые T секунд для вычисления очередного значения Y1 поступает новая порция исходных данных: A, B, C, D - и каждые T секунд процессор 1 и процессор 2 параллельно выполняют операции умножения, а процессор 3 в это время складывает произведения, полученные в предыдущие T секунд. Очевидно, что с увеличением массива исходных данных эффект от распараллеливания стремится к предельному значению. Более того, почти такой же эффект получается при вычислении Y2. Правда, здесь каждая порция исходных данных будет преобразовываться в результат уже в течение 3T секунд, зато в работе одновременно будут находиться не 2, а 3 порции исходных данных. Распараллеливание вычислений по потоку данных лежит в основе так называемых конвейерных вычислений и часто позволяет добиться большего эффекта, чем распараллеливание, основанное на декомпозиции алгоритма. Поэтому там, где это возможно, его желательно использовать в первую очередь.

Модели параллельных вычислений

Параллельное программирование представляет дополнительные источники сложности - необходимо явно управлять работой тысяч процессоров, координировать миллионы межпроцессорных взаимодействий. Для того решить задачу на параллельном компьютере, необходимо распределить вычисления между процессорами системы, так чтобы каждый процессор был занят решением части задачи. Кроме того, желательно, чтобы как можно меньший объем данных пересылался между процессорами, поскольку коммуникации значительно больше медленные операции, чем вычисления. Часто, возникают конфликты между степенью распараллеливания и объемом коммуникаций, то есть чем между большим числом процессоров распределена задача, тем больший объем данных необходимо пересылать между ними. Среда параллельного программирования должна обеспечивать адекватное управление распределением и коммуникациями данных.

Из-за сложности параллельных компьютеров и их существенного отличия от традиционных однопроцессорных компьютеров нельзя просто воспользоваться традиционными языками программирования и ожидать получения хорошей производительности. Рассмотрим основные модели параллельного программирования, их абстракции адекватные и полезные в параллельном программировании:

Процесс/канал (Process/Channel)

В этой модели программы состоят из одного или более процессов, распределенных по процессорам. Процессы выполняются одновременно, их число может измениться в течение времени выполнения

программы. Процессы обмениваются данными через каналы, которые представляют собой однонаправленные коммуникационные линии, соединяющие только два процесса. Каналы можно создавать и удалять.

Обмен сообщениями (Message Passing)

В этой модели программы, возможно различные, написанные на традиционном последовательном языке исполняются процессорами компьютера. Каждая программа имеют доступ к памяти исполняющего её процессора. Программы обмениваются между собой данными, используя подпрограммы приема/передачи данных некоторой коммуникационной системы. Программы, использующие обмен сообщениями, могут выполняться только на *MIMD* компьютерах.

Параллелизм данных (Data Parallel)

В этой модели единственная программа задает распределение данных между всеми процессорами компьютера и операции над ними. Распределяемыми данными обычно являются массивы. Как правило, языки программирования, поддерживающие данную модель, допускают операции над массивами, позволяют использовать в выражениях целые массивы, вырезки из массивов. Распараллеливание операций над массивами, циклов обработки массивов позволяет увеличить производительность программы. Компилятор отвечает за генерацию кода, осуществляющего распределение элементов массивов и вычислений между процессорами. Каждый процессор отвечает за то подмножество элементов массива, которое расположено в его локальной памяти. Программы с организацией параллельных вычислений данных могут быть оттранслированы и исполнены как на *MIMD*, так и на *SIMD* компьютерах.

Общая память (Shared Memory)

В этой модели все процессы совместно используют общее адресное пространство. Процессы асинхронно обращаются к общей памяти как с запросами на чтение, так и с запросами на запись, что создает проблемы при выборе момента, когда можно будет поместить данные в память, когда можно будет удалить их. Для управления доступом к общей памяти используются стандартные механизмы синхронизации - семафоры и блокировки процессов.

Среда выполнения

Среда выполнения с организацией параллельных вычислений обеспечивает единообразие и предсказуемость выполняемых одновременно приложений и компонентов приложений. Два примера преимуществ среды выполнения с организацией параллельных вычислений - совместное планирование задач и совместное блокирование. Среда выполнения с организацией параллельных вычислений использует совместный планировщик задач, реализующий алгоритм переноса нагрузки, для эффективного распределения нагрузки на вычислительные ресурсы. Например, представим приложение, в котором имеется два потока, управляемые одной средой выполнения. Если один поток завершает запланированную для него задачу, он может принять на себя часть работы другого потока. Этот механизм позволяет уравновешивать общую рабочую нагрузку на приложение.

Среда выполнения с организацией параллельных вычислений также предоставляет примитивы синхронизации, использующие совместную блокировку для синхронизации доступа к ресурсам. Например, рассмотрим задачу, для которой необходимо обеспечить эксклюзивный доступ к общему ресурсу. Совместная блокировка позволяет использовать в среде выполнения оставшийся такт для выполнения другой задачи, пока первая задача ожидает освобождения ресурса. Этот механизм обеспечивает максимально эффективное использование вычислительных ресурсов.

Архитектура

Среда выполнения с организацией параллельных вычислений разделена на четыре компонента: библиотека параллельных шаблонов (PPL), библиотека асинхронных агентов, планировщик заданий и диспетчер ресурсов. Эти компоненты распределены между операционной системой и приложениями. На следующем рисунке показано, как компоненты среды выполнения с организацией параллельных вычислений взаимодействуют в операционной системе и приложениях.

Среда выполнения с организацией параллельных вычислений предоставляет широкие возможности комбинирования, т.е. позволяет сочетать имеющиеся функциональные возможности для выполнения дополнительных задач. Среда выполнения с организацией параллельных вычислений поддерживает множество функций (например, параллельные алгоритмы), унаследованных от компонентов более низкого уровня.

Среда выполнения с организацией параллельных вычислений также предоставляет примитивы синхронизации, использующие совместную блокировку для синхронизации доступа к ресурсам.

Библиотека параллельных шаблонов

Библиотека параллельных шаблонов (PPL) предоставляет контейнеры и алгоритмы общего назначения для выполнения специфических операций параллелизма. *PPL* обеспечивает поддержку императивного параллелизма данных, предоставляя параллельные алгоритмы, которые распределяют вычислительные ресурсы между вычислениями коллекций или наборов данных. Кроме того, библиотека поддерживает параллелизм задач, предоставляя объекты задач, распределяющие вычислительные ресурсы между несколькими независимыми операциями.

Библиотеку параллельных шаблонов рекомендуется использовать при наличии локальных вычислений, требующих параллельного выполнения. Например, можно использовать алгоритм *Concurrency::parallel_for* для преобразования существующего цикла *for* для параллельной работы.

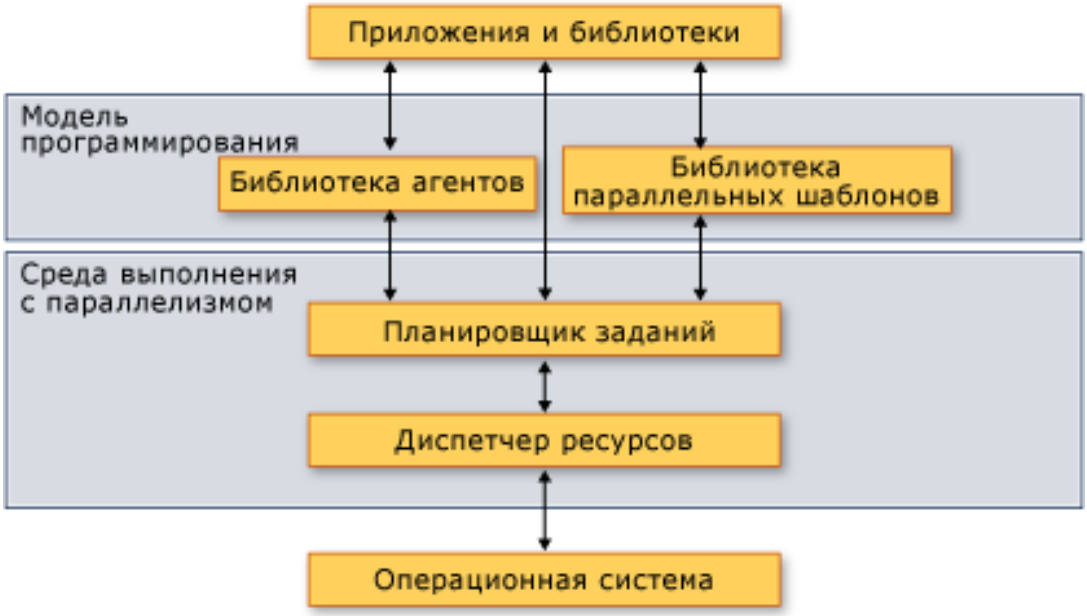

Рис. 1. *Архитектура параллельных вычислений*

Библиотека асинхронных агентов

Библиотека асинхронных агентов (или просто Библиотека агентов) предоставляет модель программирования на основе субъектов и интерфейсы передачи сообщений для недетализированного потока данных и задач по конвейеризации. Асинхронные агенты позволяют эффективно использовать задержку, выполняя работу, пока другие компоненты ожидают данных.

Рекомендуется использовать библиотеку агентов, если имеется несколько асинхронно взаимодействующих друг с другом сущностей. Например, можно создать агент, который считывает данные из файла или сетевого подключения, а затем использует интерфейсы передачи сообщений для отправки этих данных другому агенту.

Планировщик заданий

Планировщик заданий планирует и координирует задачи во время выполнения. Планировщик заданий поддерживает совместную работу и использует алгоритм переноса нагрузки для максимально эффективного использования ресурсов для обработки. Среда выполнения с организацией параллельных вычислений предоставляет планировщик по умолчанию, что устраняет необходимость в управлении подробностями инфраструктуры. Однако для удовлетворения качественных требований приложения можно использовать собственную политику планирования или связать конкретные планировщики с конкретными задачами.

Таким образом, главным критерием качества распараллеливания вычислений является сокращение общего времени решения задачи. На основе параллельных вычислений разработано параллельное программирование, но оно представляет дополнительные источники сложности - необходимо явно управлять работой тысяч процессоров, координировать миллионы межпроцессорных взаимодействий.

Список литературы

1. **Ильюшин Ю. В.** Проектирование распределенной системы со скалярным воздействием // Научное обозрение. М., 2011. № 4. С. 85-90.
2. **Ильюшин Ю. В.** Проектирование системы управления температурными полями туннельных печей конвейерного типа // Научно-технические ведомости СПбГПУ. Серия «Информатика. Телекоммуникации. Управление». 2011. № 3 (126). С. 67-72.
3. **Ильюшин Ю. В., Чернышев А. Б.** Определение шага дискретизации для расчета теплового поля трехмерного объекта управления // Известия Южного федерального университета. Таганрог, 2011. № 6. С. 192-200.
4. **Ильюшин Ю. В., Чернышев А. Б.** Устойчивость распределенных систем с дискретными управляющими воздействиями // Известия Южного федерального университета. Таганрог, 2010. № 12. С. 166-171.
5. **http://parallel.ru/vvv/lec7.html**