Программная модель параллельной реализации метода ветвей и границ

А.Л. Фомин

***Аннотация*—В данной работе предложена программная модель параллельной реализации метода ветвей и границ. Созданный на ее основе симулятор позволяет исследовать работу реального приложения в различных режимах. Принятый подход предполагает, что параллельное выполнение процессов реального приложения моделируется посредством последовательного выполнения псевдопараллельных процессов симулятора. Для их синхронизации применяются временные метки событий.**

***Ключевые слова*—метод ветвей и границ, параллелизм, моделирование параллельной системы.**

I. Введение

Метод ветвей и границ — один из подходов к решению задач оптимизации, который нередко реализуется с использованием параллелизма. Однако круг всех возможных способов такой реализации, даже в пределах одного подхода, может быть практически необозрим. При этом исследование и сравнение различных способов в реальном решении задач связано со значительными затратами времени и вычислительных ресурсов. Все это приводит к идее провести подобное исследование путем имитационного моделирования.

II. Предварительные замечания

A. Общая схема метода ветвей и границ

Метод ветвей и границ предполагает разбиение исходной задачи на подзадачи с исключением тех из них, которые заведомо не позволяют получить оптимального решения. Для применения метода задается условие отсева задач и правило разбиения задачи на подзадачи.

На каждом шаге метода имеется коллекция подзадач исходной задачи, перед началом работы в ней содержится только сама исходная задача.

Последовательность действий для отдельного шага такова:

1. выбрать и удалить из коллекции подзадачу;
2. проверить для нее условие отсева;
3. если оно не выполнено, применить к ней правило разбиения и добавить в коллекцию полученные новые подзадачи.

Работу метода можно представить в виде постепенного построения дерева подзадач: его вершины представляют все построенные задачи, а дуги соединяют задачу с подзадачами, полученными путем ее декомпозиции. Коллекция подзадач состоит из всех терминальных вершин дерева.

При параллельной реализации шаги метода параллельно применяются к различным подзадачам.

B. Краткое описание библиотеки bnb-solver

Библиотека bnb-solver (подробности о ней содержатся в работах [1, 3]) предоставляет параллельную реализацию метода ветвей и границ. При решении задачи с ее применением запускается ряд процессов, и в каждом из них создаются объекты: решатель и балансировщик. Решатель непосредственно выполняет шаги метода ветвей и границ, а балансировщик управляет его работой. Решатели из различных процессов взаимодействуют путем обмена сообщениями посредством MPI.

На каждом шаге решатель сообщает балансировщику о своем состоянии, а тот в ответ выдает команду (ее могут сопровождать дополнительные параметры), которую решатель должен выполнить.

Основные виды команд таковы:

1. строить продолжение дерева подзадач;
2. отправить сообщение;
3. ожидать сообщения;
4. закончить работу.

Точный список команд и их параметры зависят от типа выбранного балансировщика.

Состояние решателя изменяется как итог выполнения команды. После этого делается следующий шаг (кроме очевидного случая: последняя команда завершает весь процесс).

На каждом шаге все терминальные вершины дерева подзадач распределены между решателями. Новые вершины, которые появляются при разрастании дерева, изначально принадлежат тому же решателю, что их родительская вершина. Однако вершины также могут пересылаться между решателями. Назначение балансировщика — используя эту возможность, обеспечить равномерное распределение нагрузки между решателями.

В данной работе решается задача создания симулятора для моделирования работы приложения с bnb-solver, имеющего значительное число (несколько сотен тысяч) параллельных процессов. При разработке и использовании приложений с bnb-solver такой симулятор существенно упрощает выбор алгоритма балансировки и уточнение его дополнительных параметров.

C. Существующие подходы к моделированию

Использование программных моделей для

Статья получена 21 октября 2015. Работа выполнена при поддержке гранта РФФИ 13-07-00768 А и представляет собой результат выпускной квалификационной работы на степень бакалавра.

А.Л. Фомин АО “МЦСТ” Москва, Россия (e-mail: fin-al@yandex.ru).

исследования параллельных реализаций метода ветвей и границ не ново: одна из первых подобных моделей предложена в работе [2]. В ней описана реализация симулятора, работа которого может служить моделью работы реального приложения со множеством параллельных процессов. Для представления каждого процесса приложения запускается новый процесс симулятора, и все эти процессы взаимодействуют с помощью именованных каналов. Подобный подход, очевидно, предполагает создание большого количества объектов ОС, поэтому при запуске симулятора на ПК невозможно добиться приемлемого времени моделирования уже для нескольких сотен процессов.

Принятый автором подход предполагает полный отказ от параллелизма в самом симуляторе: параллельная работа процессов реального приложения моделируется в пределах единственного системного процесса.

III. Устройство и работа симулятора

Симулятор предназначен для исследования работы балансировщика. Он моделирует решение задачи с помощью bnb-solver и составляет трассу всех шагов, в которой отражаются состояния решателей и выполненные команды для всех процессов.

В основе симулятора лежит набор модельных процессов, которые представляют параллельные процессы приложения с bnb-solver. Однако при работе симулятора, в отличие от реального приложения, процессы выполняют свои шаги по очереди, а не параллельно. Это требует введения модельного времени, которое отличается от реального, и на модельные процессы ложится задача его учета. Общее время решения, которое используется для оценки балансировщика, определяется как максимум модельного времени, затраченного каждым модельным процессом.

Модельный процесс непосредственно управляет действиями следующих основных подсистем:

1. балансировщик: определяет последовательность команд для процесса;
2. решатель: моделирует решение задачи;
3. коммуникатор: отвечает за обмен сообщениями между процессами.

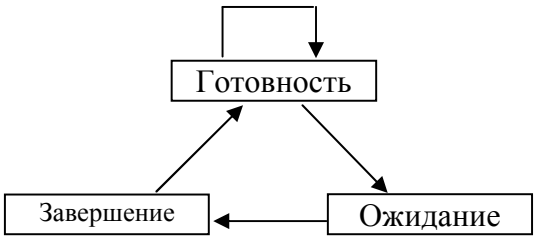
Каждый модельный процесс располагает собственным балансировщиком и решателем, но все процессы взаимодействуют с одним коммуникатором.

Основное свойство решателя симулятора — полная совместимость с любым возможным балансировщиком из bnb-solver. Однако при его реализации все действия по решению задачи выполняются условно. В отличие от реального приложения, единственная задача симулятора — составление трассы, в которой содержится последовательность команд, выполненных каждым процессом. Поэтому необходимо только определять, какое время затрачивается на выполнение команды и как изменяется при этом состояние всех подсистем симулятора. Этого возможно добиться без фактического выполнения самих команд, что позволяет радикально сократить затраты времени на моделирование по сравнению с реальной работой приложения с bnb-solver.

IV. Моделирование отдельного шага процесса

A. Состояния и общий порядок действий процесса

Перед выполнением каждого шага для модельного процесса определено его состояние: готовность, ожидание или завершение ожидания (смысл последнего состояния станет ясен при обсуждении моделирования обмена сообщениями). Возможные переходы состояний показаны ниже:



При создании для модельного процесса всегда устанавливается состояние готовности. Процесс выполняет команды балансировщика только в этом состоянии и возвращается в него, как только выполнение команды завершается.

Порядок действий при выполнении каждого шага таков:

1. запросить у решателя сведения о его состоянии;
2. передать их балансировщику и получить команду;
3. выполнить команду.

По команде завершения работы модельный процесс выводит в выделенный поток (общий для всех процессов) итоговые сведения о своей работе, и после этого существование процесса прекращается.

Далее описан порядок выполнения основных команд, которое может задействовать решатель и коммуникатор.

B. Моделирование решения задачи

Команда, которая предписывает строить продолжение дерева подзадач, полностью исполняется решателем, и процесс передает ему все необходимые параметры команды, полученные от балансировщика.

Перед выполнением команды решатель располагает некоторой коллекцией терминальных вершин дерева подзадач. Выполняя команду, он повторяет в цикле следующие действия.

1. Выбрать и удалить из коллекции какую-либо вершину. Дополнительные параметры команды, полученные от балансировщика, могут уточнять, какие вершины следует выбирать в первую очередь.
2. Проверить условие отсева для выбранной вершины.
3. Если оно не выполняется, применить к этой вершине правило разбиения и заменить ее в коллекции на полученное множество дочерних вершин.

Балансировщик может ограничивать число итераций цикла или заставить выполнять его до момента, когда

коллекция вершин станет пустой.

При реальном решении задачи с каждой вершиной дерева подзадач связаны дополнительные данные, но в симуляторе они сводятся к единственному числу h — высоте вершины. Проверка условия отсева и декомпозиция проводятся с помощью случайного эксперимента. Для этого задается априорная максимальная высота дерева $H > 0$. Если терминальная вершина имеет высоту h, то при моделировании принимается, что с вероятностью h/H для данной вершины условие отсева выполняется, а с вероятностью $1 - h/H$ вместо нее появляются 2 новые дочерние вершины высоты $h + 1$. При этом, если $h = H$, то условие отсева выполняется в любом случае, поэтому рост дерева не может продолжаться бесконечно. Для различных вершин эксперименты считаются независимыми.

Выполнив команду, решатель сообщает процессу число сделанных итераций цикла. Эти сведения используются для учета затраченного модельного времени: оно принимается пропорциональным числу итераций.

C. Моделирование отправки сообщения

В команде отправки сообщения балансировщик указывает его содержание и процесс-получатель. Далее приведена последовательность действий процесса, который выполняет эту команду.

1. Подготовить данные, которые составят содержание сообщения. Для простых сообщений процесс способен проделать это, не обращаясь к остальным подсистемам симулятора. Однако основной вид сообщений, который используется для перераспределения нагрузки между процессами, — это пакет терминальных вершин дерева подзадач. Подготовка такого сообщения, очевидно, задействует решатель, который извлекает часть вершин из своей коллекции и создает из них пакет для получателя.
2. Учесть затраты модельного времени на подготовку сообщения.
3. Вызвать коммуникатор, которому передается само сообщение, его отправитель и получатель, а также временная метка — текущее значение модельного времени у отправителя. Заметим, что в библиотеке bnb-solver временные метки не используются, но для симулятора они критически важны. Их предназначение описано в разделе о работе коммуникатора. Идея использования временных меток заимствована из [4].

В симуляторе, как и в библиотеке bnb-solver, отправка сообщения не заставляет процесс-отправитель дожидаться его доставки: коммуникатор хранит копии всех сообщений.

D. Моделирование получения сообщения

В команде получения сообщения балансировщик указывает его отправителя, предписывая ожидать сообщения от определенного процесса или от произвольного. В отличие от отправки, получение сообщения может заблокировать процесс в ожидании, если сообщение еще не отправлено. В симуляторе блокировка реализована как переход процесса в особое состояние ожидания.

Далее приведена последовательность действий процесса, который выполняет команду получения сообщения.

1. Вызвать коммуникатор, которому передается ссылка на буфер для сообщения, его получатель и отправитель (определенный или произвольный).
2. Новое состояние процесса зависит от ответа коммуникатора на запрос.
   (a) Если сообщение уже готово, коммуникатор доставляет его получателю (копирует в буфер), при этом указывая его отправителя и время доставки. Последнее определяется на основе временной метки, которой снабжает сообщение отправитель. Получатель остается в состоянии готовности.
   (b) Если сообщение еще не готово, коммуникатор извещает об этом процесс, и тот переходит в состояние ожидания. При этом шаг процесса прекращается. Впоследствии, как только появляется подходящее сообщение, коммуникатор доставляет его получателю. Об этом событии он уведомляет процесс самостоятельно, передавая при этом отправителя сообщения и время доставки. Получив подобное уведомление, получатель переходит в состояние завершения ожидания.

Дальнейшие действия процесса в обоих случаях одинаковы.

3. Используя время доставки, установить свое новое модельное время как максимум из текущего и времени доставки.
4. Разобрать принятое сообщение. Порядок действий при разборе определяется типом сообщения, который всегда передается вместе с ним. Процесс в состоянии завершения ожидания при этом возвращается в состояние готовности.
5. Учесть затраты модельного времени на разбор сообщения.

Процесс использует состояние завершения ожидания как признак того, что на своем очередном шаге он должен выполнить действия по разбору принятого сообщения, а не запрашивать очередную команду балансировщика. При реальной работе приложения с bnb-solver разбор доставленного сообщения выполняется на одном шаге с вызовом коммуникатора, но этот шаг может содержать период блокировки процесса в ожидании сообщения. Однако в симуляторе процессы выполняют шаги последовательно, а не параллельно, поэтому блокировка процесса реализована

как прекращение текущего шага и переход в состояние ожидания. По этой причине при запросе на получение сообщения коммуникатор должен возвращать управление процессу, даже если сообщения еще нет. В последнем случае полное выполнение команды потребует дополнительного шага.

V. Принципы работы коммуникатора

Основные способы взаимодействия процессов с коммуникатором — запросы на отправку и получение сообщения. Если запрос невозможно обслужить немедленно, коммуникатор запоминает его и может завершить его обслуживание позднее, при поступлении нового запроса.

При запросе на отправку коммуникатору передается само сообщение, его отправитель и получатель, а также время отправки. Далее приведен порядок действий коммуникатора при обработке запроса на отправку сообщения.

1. Проверить, есть ли ожидающий обслуживания запрос на получение сообщения, исходящий от заданного получателя. Если в последнем запросе указан какой-то определенный отправитель, сверить, совпадает ли он с процессом, от которого исходит запрос на отправку.
2. Далее выполняется одно из следующих действий.
   (a) Если есть процесс, который ожидает это сообщение, доставить его получателю и известить его об этом событии, тем самым выводя его из состояния ожидания. При этом коммуникатор передает получателю и время доставки сообщения, позволяя определить затраты модельного времени на ожидание.
   (b) Если запросов на получение этого сообщения нет, сохранить его копию для будущей доставки. Вместе с сообщением запоминается его отправитель и время доставки, которое коммуникатор передаст будущему получателю.

Время доставки сообщения коммуникатор определяет, используя время отправки: к нему прибавляется дополнительное время, что моделирует задержку при передаче данных по каналу связи. В симуляторе эта модельная задержка принимается равной $L + S/B$, где L — постоянная задержка, которая не зависит от длины сообщения, S — длина переданного сообщения, B — пропускная способность канала связи, то есть максимальная длина сообщения, которое можно передать за одну единицу модельного времени. Постоянная задержка и пропускная способность канала предполагается одинаковой при всех взаимодействиях процессов.

Длина сообщения, которая используется при определении задержки, не представляет его реальную длину, а назначается сообщению условно. Для всех возможных компонентов сообщений: подзадач, а также служебных уведомлений, — вводится их условная длина, и за длину всего сообщения принимается суммарная условная длина его компонентов. Это позволяет определять задержки при передаче данных, исходя из их объема при реальной работе приложения с bnb-solver, при том, что сообщения, используемые в симуляторе, имеют совершенно иное содержание.

При запросе на получение коммуникатору передается ссылка на буфер для сообщения, его получатель и отправитель, при этом получатель может заявить о готовности принять сообщение от любого отправителя. Далее приведен порядок действий коммуникатора при поступлении обработке на получение.

1. Проверить, есть ли ожидающие доставки сообщения, предназначенные для получателя. Если в запросе указан какой-то определенный отправитель, учитываются только сообщения от него.
2. Далее выполняется одно из следующих действий.
   (a) Если сообщение готово, доставить его получателю, попутно передав и время доставки. Из всех подходящих сообщений коммуникатор всегда выбирает то, у которого время доставки самое раннее.
   (b) Если подходящее сообщение еще не готово, запомнить запрос, а также известить получателя, что тот должен перейти в состояние ожидания.

VI. Управление процессами в целом

Перед запуском моделирования создается пул модельных процессов. Работа симулятора в целом сводится к выполнению в цикле следующих действий.

1. Опросить все процессы из пула, которые сообщают свое состояние и текущее модельное время. Процесс в состоянии завершения ожидания, то есть тот, у которого есть доставленное коммуникатором сообщение, но еще не выполнена его обработка, разбирает свое сообщение и переходит в состояние готовности. Затраты времени на этот разбор увеличивают его текущее время.
2. Из всех процессов в состоянии готовности выбрать один из тех, у которых текущее время минимально, и позволить ему выполнить одну команду балансировщика. После этого процесс сообщает, была ли выполнена команда завершения работы.
3. Если работа этого процесса завершена, удалить его из пула.

Цикл симулятора прекращается, как только пул процессов становится пуст, то есть все они выполнили команду завершения.

Подобный порядок действий гарантирует, что все запросы процессов к коммуникатору на отправку сообщения происходят строго по возрастанию их текущего времени. Для оправдания этого требования представим, что при работе симулятора происходит

последовательность событий:

1. процесс S_1 с текущим временем t_1 вызывает коммуникатор для отправки сообщения процессу R;
2. процесс R вызывает коммуникатор для получения сообщения от произвольного отправителя;
3. процесс S_2 с текущим временем t_2 вызывает коммуникатор для отправки сообщения процессу R.

В этом случае коммуникатор доставит получателю R сообщение от S_1, поскольку при поступлении запроса от R коммуникатору еще не известно сообщение от S_2. Но если $t_2 < t_1$, то при параллельном выполнении процессов запрос от S_2 опережает во времени запрос от S_1, и при поступлении запроса от S_1 коммуникатор уже должен получить и запрос от S_2. Таким образом, подобная последовательность событий делает модель неадекватной. Введенное требование позволяет исключить ее появление: при $t_2 < t_1$ процесс S_2 получит возможность сделать запрос прежде S_1.

VII. Заключение

На основе предложенной программной модели разработан симулятор, который внедрен в научно-исследовательскую работу Центра распределенных вычислений ИППИ РАН. Дальнейшее уточнение модели возможно путем использования более сложной модели передачи сообщений с учетом структуры коммуникационной сети.

Благодарности

Автор благодарит М.А. Посыпкина за неоценимую помощь в создании этой работы.

Библиография

[1] Посыпкин М.А. Решение задач глобальной оптимизации в среде распределенных вычислений. Программные продукты и системы, 2010, т. 1, с. 23 – 29.

[2] Arie de Bruin, Alexander H.G. Rinnooy Kan, Harry W.J.M. Trienekens. A simulation tool for the performance evaluation of parallel branch and bound algorithms. Mathematical Programming, April 1988, vol. 42, issue 1 – 3, pp. 245 – 271.

[3] Evtushenko Y., Posypkin M., Sigal I. A framework for parallel large-scale global optimization. Computer Science – Research and Development, 2009, vol. 23, issue 3 – 4, pp. 211 – 215.

[4] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. Communication of the ACM, July 1978, vol. 21, issue 7, pp. 558 – 565.

A software simulation tool for the parallel branch and bound method implementation.

A.L. Fomin

***Abstract*—This paper describes a software simulation tool for the parallel branch and bound method implementation. Such a tool allows to analyse performance of the implementation in various modes. According to the approach chosen, parallel execution of the application processes is modelled by sequential execution of the pseudoparallel simulator processes. To synchronize them, the timestamps of events are involved.**

***Keywords*—branch and bound, parallel computing, simulation.**