Gem #154: Многоядерное прохождение лабиринта, Часть 2

Автор: Pat Rogers - AdaCore

Краткое содержание: В данной серии публикаций описывается проект параллельного разрешения лабиринтов («amazing»), включенный в примеры GNAT Pro. В первом Gem серии был представлен сам проект и проектировочный подход к параллельному программированию. Второй Gem серии содержит разъяснение основных изменений, которые необходимы для обеспечения оптимальной производительности в многоядерных архитектурах. Данные изменения касаются критической проблемы производительности, которая не была известна во время первого выпуска программы в 1980 г. Они демонстрируют фундаментальные отличия традиционного многопроцессового и современного многоядерного программирования.

Давайте начнём…

Эта серия статей Gem описывает параллельный проект Maze solver ("удивительный лабиринт"), включенный в примеры GNAT Pro. Первый, Gem #153, в серии статей представил сам проект и объяснил подход к параллельному программированию. Этот второй Gem #154 исследует основное изменение, которое было необходимо для оптимальной производительности на многоядерных архитектурах. Это изменение решило критическое узкое место производительности, которого не было, когда оригинальная программа была впервые развернута в 1980-х годах, иллюстрируя одно из фундаментальных различий между программированием традиционной многопроцессорной аппаратной платформой и современной многоядерной аппаратной платформой.

Исходной целевой машиной был Sequent Balance 8000, симметричный мультипроцессор с восемью процессорами и общей памятью. Операционная система прозрачно отправляла задачи Ada процессорам, поэтому для нее можно было написать очень портативную параллельную программу Ada. В 1980-х годах это была очень привлекательная машина, как вы можете себе представить. Полученная программа успешно продемонстрировала способность Ada использовать такие архитектуры, а также общие преимущества параллельного выполнения. В частности, время выполнения последовательной версии решателя лабиринта росло тревожными темпами по мере увеличения числа решений лабиринта, в то время как параллельная версия демонстрировала лишь незначительное увеличение. (Помните, что дело в том, чтобы найти все возможные решения для данного лабиринта, а не только один.)

Программа действительно была очень портативной и работала на нескольких машинах разных производителей, как параллельных, так и нет. Со временем мы включили достижения языковых редакций, в первую очередь защищенные типы, и добавили такие функции, как переключатели командной строки для гибкости, но архитектура и реализация в основном остались неизменными. То есть до недавнего времени.

Как описано в первой Gem в этой серии, программа "наводняет" лабиринт с задачами искателя в классическом дизайне "разделяй и властвуй", каждый искатель ищет выход из заданной начальной точки. Самый первый искатель, конечно, начинает с входа в лабиринт, но поскольку любая задача искателя сталкивается с пересечениями в лабиринте, он назначает другую идентичную задачу каждому альтернативному местоположению, сохраняя одну для себя. Таким образом, задача поиска, которая находит выход, обнаружила только часть полного пути решения через лабиринт. Если бы самый первый искатель нашел выход, у него было бы полное решение, но все остальные искатели имеют только часть любого данного пути решения, потому что они не начинали со входа.

Когда искатели пересекают лабиринт, они отслеживают места лабиринта, которые они посещают, чтобы эти места могли отображаться, если выход в конечном итоге найден. Но, как мы видели, эти места составляют лишь частичный путь через лабиринт. Поэтому, когда успешный искатель отображает все решение, он также должен знать местоположения решения до своей собственной начальной точки, а также местоположения, которые он пересек сам, чтобы добраться до

выхода. Для выполнения этого требования при инициировании поиска в данном начальном местоположении ему также предоставляется текущее решение, известное до этого местоположения. Самому первому искателю просто дается пустое решение, известное как "след" в программе. Успешные поисковики отображают как часть, которую они обнаружили, так и часть, которую они получили при запуске.

Обратите внимание, что эти частичные решения потенциально являются общими, в зависимости от лабиринта. (Решения являются уникальными, если какие-либо составляющие местоположения лабиринта различны, но это не исключает частичного совместного использования.) Вероятно, эти местоположения лабиринта, расположенные ближе к входу, будут в значительной степени разделены между большим количеством уникальных решений. Концептуально, полные решения образуют дерево последовательностей местоположений, причем ранее общие сегменты появляются в дереве раньше, а уникальные подсегменты появляются под ними. Вход в лабиринт появляется один раз, в корне наверху дерева, тогда как выход из лабиринта появляется в конце каждого решения.

Представьте себе, как можно представить это дерево. Учитывая, что сегменты решений - следы - вероятно, разделены логически, возможно, мы также можем разделить их физически. Тем не менее, как общая структура данных, гоночные условия являются очевидной проблемой. Поэтому мы хотим представление, которое минимизирует блокировку, необходимую для взаимного исключения. Мы также хотим представление, которое может содержать любое количество пар местоположений на сегмент, потому что лабиринты изначально генерируются случайным образом. То есть мы не знаем, сколько локаций будет содержать данное решение, тем более, сколько будет решений.

Неограниченный, динамически распределяемый список местоположений лабиринта прекрасно отвечает этим целям. Он может непосредственно представлять логическое разделение и может обрабатывать трейлы любой длины, пока доступно достаточно памяти. Более того, блокировка взаимного исключения не требуется, поскольку нам нужно только добавить сегменты списка к предыдущим, существующим сегментам. Нет необходимости изменять сами предыдущие сегменты, поэтому вообще не нужно блокировать дерево!

Представление кажется идеальным, и для исходной симметричной многопроцессорной задачи это был разумный подход, но когда программа работала на современных многоядерных машинах, производительность была очень низкой. Действительно, использование отдельных процессоров было настолько низким, что последовательная версия решателя лабиринтов была вполне конкурентоспособна с параллельной версией.

Плохое использование процессора является ключом к проблеме. Несмотря на то, что мы используем несколько процессоров и можем иметь столько потоков, сколько нам нужно, отдельные процессоры работают плохо. Проблема вызвана плохим использованием кеша, что само по себе является результатом плохой локальности ссылок. В частности, динамически распределяемые элементы в цепочках не находятся в местах памяти, достаточно близких друг к другу, чтобы находиться в одной и той же строке кэша, тем самым вызывая много пропусков кэша и низкую общую производительность процессора.

Проблема заключается в том, что задачи поисковика должны также проверять местоположения в своих предыдущих путях решения при поиске выхода. (Другими словами, не только при отображении решений.) Они делают это для предотвращения ложных круговых решений через лабиринт, что стало возможным благодаря наличию пересечений. Поэтому задачи поисковика должны определить, были ли они в заданном месте в лабиринте, прежде чем включать это место в свое решение. Однако для выполнения этой проверки не нужно посещать все пары местоположений в следе. Наличие пересечения в предшествующем сегменте пути достаточно для указания кругового движения, поэтому каждый след включает в себя список пересечений, и именно этот вторичный список проверяют искатели. К сожалению, любые преимущества этой оптимизации реализации перекрываются результатами промахов кэша.

Другое представление трассы необходимо для программ, предназначенных для многоядерных целей, с гораздо лучшей локализацией. Массивы имеют эту точную характеристику, поэтому мы выбрали ограниченный список на основе массива для представления трасс. Этот выбор не удивит тех, кто знаком с этой проблемой, даже несмотря на то, что полученное в результате копирование и отсутствие физического обмена будет оспаривать это.

В следующем Gem этой серии мы предоставим подробную информацию об этом изменении реализации и задействованных повторно используемых компонентах.

Как уже упоминалось, проект «удивительный лабиринт» поставляется совместно с компилятором GNAT Pro. Найдите его в каталоге `share/examples/gnat/amazing/`, расположенном в корневом каталоге вашего компилятора. Обратите внимание, что описанное изменение дизайна появится в следующих выпусках компилятора.

Связанный со статьёй текст программы

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Автор: Pat Rogers - AdaCore

Пэт Роджерс является специалистом по вычислительной технике с 1975 года, в основном работая над микропроцессорными приложениями реального времени на языках Ada, C, C ++ и других, включая высокоточные имитаторы полета и системы диспетчерского контроля и сбора данных (SCADA), контролирующие опасные материалы. Впервые изучив Аду в 1980 году, он был директором лаборатории Ada9X для Совместной программы передовых технологий удара ВВС США (Ada9X Laboratory for the U.S. Air Force’s Joint Advanced Strike Technology Program), главным исследователем (Principle Investigator) в распределенных системах и исследовательских проектах по отказоустойчивости с использованием Ada для ВВС и армии США (U.S. Air Force and Army), а также заместителем директора (Associate Director) по исследованиям. в исследовательском центре разработки программного обеспечения НАСА (Research at the NASA Software Engineering Research Center). У него есть Б.С. и М.С. степени в области компьютерных систем и компьютерных наук в Университете Хьюстона (the University of Houston) и степень доктора философии (Ph.D) в области компьютерных наук из Университета Йорка, Англия (the University of York, England). Как старший технический специалист (the Senior Technical Staff) AdaCore, он специализируется на поддержке разработчиков встраиваемых систем в режиме реального времени, создает и проводит учебные курсы, а также является руководителем проекта и разработчиком (project leader and a developer) подключаемого модуля GNATbench Eclipse для Ada. Он также имеет черный пояс 3-го дана в Таэквондо и является основателем клуба AdaCore «Злые дяди» (“The Wicked Uncles”).

Last Updated: 10/13/2017
Posted on: 10/9/2013

Обсуждение…