Gem #153: Многоядерное прохождение лабиринта, Часть 1

Автор: Pat Rogers - AdaCore

Краткое содержание: В данном Gem представляется проект «amazing», включенный в примеры компилятора GNAT Pro. Название проекта связано с тем, что его задача - нахождение выхода из лабиринтов (англ. «maze» - лабиринт). Но это - не обычные лабиринты, в которых есть только один выход. Решений может быть множество, например, десятки тысяч. Суть в том, чтобы найти все решения как можно быстрее. Поэтому эти задачи решаются параллельно, с применением нескольких ЦП и проектирование методом «разделяй и властвуй». В первой публикации данной серии мы представим программу и разъясним подход к разрешению данной проблемы.

Давайте начнём…

Эта серия Gem представляет проект "удивительный лабиринт" ("amazing"), включенный в примеры компилятора GNAT Pro. Проект назван так потому, что он включает в себя решение лабиринта (как в "Джо и Джули проходят лабиринт" - "Joe and Julie go a-mazing"). Но это не типичные лабиринты, которые имеют только одно решение. Эти лабиринты могут иметь множество решений, десятки тысяч, например. Дело в том, чтобы найти их все как можно быстрее. Поэтому мы решаем лабиринты одновременно - паралельно, применяя несколько процессоров в дизайне "разделяй и властвуй". В этом первом Gem мы представляем программу и объясняем подход.

На самом деле у нас есть две программы для решения лабиринта: одна последовательная и одна параллельная. Основываясь на понятии мыши, решающей лабиринт, последовательная программа называется мышью, а параллельная версия – вы догадались – мышами. Оба вызываются из командной строки с необходимые и необязательные параметры. Доступные переключатели несколько различаются между ними, но в обоих случаях вы можете либо создать и решить новый лабиринт, либо повторно решить ранее созданный лабиринт.

При создании нового лабиринта, у вас есть возможность сделать его "идеальным" ("perfect"), то есть только один выход. В противном случае лабиринт будет иметь неизвестное количество решений. Для наших целей мы используем лабиринты, которые не идеальны, и на самом деле количество решений зависит исключительно от размера лабиринтов и случайного способа их генерации.

Переключатели "`-h`" и "`-w`" позволяют указать высоту и ширину нового лабиринта, но кроме этого их расположение определяется случайным образом. Кроме того, параллельная программа позволяет указать общее количество задач, доступных для решения лабиринта с помощью переключателя "`-t`". Этот переключатель полезен для экспериментов, например, при определении эффекта наличия большого количества задач или при определении оптимального количества задач относительно количества доступных процессоров. По умолчанию доступно четыре задачи. Параллельная версия программы будет работать на стольких процессорах, сколько доступно.

Наконец, вы можете управлять отображением лабиринта и его решений. На первый взгляд это может показаться странным вариантом, но их отображение делает программу сильно связанной и сериализованной, скрывая преимущества параллелизма и затрудняя определение последствий изменений дизайна. Отключение дисплея достигается с помощью переключателя"`-q`".

После того, как любая программа решает новый лабиринт, вас спрашивают, Хотите ли вы его сохранить. Если это так, вы указываете имя файла, а затем программа записывает его как поток. Чтобы повторно решить существующий лабиринт, вы указываете как переключатель "`-f`", так и имя файла.

По мере выполнения программ они отображают лабиринт, уникальные решения через лабиринт и общее количество обнаруженных решений (если не применяется переключатель "`-q`"). В

настоящее время существует два вида "консоли", поддерживаемой для отображения этой информации. Выбор определяется при построении исполняемых файлов, под управлением переменной сценария, имеющей возможные значения "Win32" и "ANSI". (Терминалы, поддерживающие escape-последовательности ANSI, распространены в системах Linux, поэтому существует широкий спектр поддерживаемых машин).

Теперь, когда вы знаете, на что способны программы, Давайте посмотрим, как они это делают.

Последовательная версия программы – из одной “мыши иллюстрирует фундаментальный подход. Когда она (условно одна “мышь”) пересекает лабиринт, она обнаруживает соединения на пути, где возможно более одного пути вперед. На самом деле, есть три пути вперед, но не четыре, потому что это потребует возвращения к только что посещенному месту. Следовательно, в любом соединении мышь сохраняет все, кроме одного из других альтернативных местоположений, вместе с потенциальным решением, как оно известно в настоящее время, а затем ищет один оставшийся отрыв. Всякий раз, когда мышь не может идти дальше - либо потому, что она столкнулась с тупиком, либо из-за того, что она нашла выход из лабиринта, - она восстанавливает одну из ранее сохраненных альтернативных пар локация/решение и продолжает оттуда. Программа завершается, когда мышь не может двигаться дальше и предыдущие альтернативы не сохраняются.

Программа `mice` использует тот же базовый подход, за исключением того, что она делает это одновременно. Тип задачи "`searcher`" реализует последовательное поведение мыши, но вместо хранения альтернатив на перекрестках он назначает новую задачу `searcher` каждой из альтернатив. Эти новые поисковики продолжают одновременно (или параллельно) с поисковиком, который назначил их, сами назначая новых поисковиков в любых точках, с которыми они сталкиваются. Только когда дополнительная задача поиска недоступна, любой поисковик сохраняет альтернативные варианты для последующего преследования. Если он восстанавливает лидерство, он использует тот же подход на любых последующих перекрестках.

Новая задача `searcher` может быть недоступна при запросе, поскольку мы используем пул экземпляров `searcher` с емкостью, управляемой параметром командной строки. Когда дальнейший прогресс невозможен, задача поиска возвращается в этот пул для последующего назначения, поэтому при поиске восстановленных версий могут быть доступны дополнительные поисковики. Основная программа ждет, пока все искатели успокоятся, ожидая в пуле заданий, прежде чем закончить.

Тело для типа задачи `Searcher` (объявленного в пакете `Search_Team`), реализующего это поведение, следующее:

```
task body Searcher is
   Path             : Traversal.Trail;
   The_Maze         : constant Maze.Reference := Maze.Instance;
   Current_Position : Maze.Position;
   Myself           : Volunteer;
   Unsearched       : Search_Leads.Repository;
begin
   loop
      select
         accept Start (My_ID : Volunteer;
                       Start : Maze.Position;
                       Track : Traversal.Trail)
         do
            Myself := My_ID;
            Current_Position := Start;
            Path := Track;
         end Start;
      or
```

```
            terminate;
         end select;

         Searching : loop
            Pursue_Lead (Current_Position, Path, Unsearched);

            if The_Maze.At_Exit (Current_Position) then
               Traversal.Threaded_Display.Show (Path, On => The_Maze);
            end if;

            exit Searching when Unsearched.Empty;

            --  Go back to a position encountered earlier that
            --  could not be delegated at the time.
            Unsearched.Restore (Current_Position, Path);
         end loop Searching;

         Pool.Return_Member (Myself);
      end loop;
   end Searcher;
```

Задача поиска сначала приостанавливается, ожидая либо инициации, либо начала преследования зацепки, либо завершения. Таким образом, рандеву предоставляет начальное местоположение и известный в настоящее время путь решения. Параметр `My_Id` является ссылкой на ту же задачу и используется задачей для возврата себя обратно в пул, когда поиск завершен. Принять тело просто копирует эти параметры в локальные переменные. Другие локальные переменные включают ссылку на сам лабиринт (для этого мы используем одиночный элемент) и хранилище неучтенных лотов, используемых для хранения пар позиция/решение для будущего поиска.

По мере того как задача ищет выход, процедура `Pursue_Lead` делегирует новые задачи поиска альтернативам при обнаружении соединений. Процедура возвращается, когда не может быть достигнут дальнейший прогресс по данному Лоту. По сути, мы "наводняем" лабиринт задачами поиска, так что этот дизайн "разделяй и властвуй", типичный для классического параллельного программирования.

В следующей Gem этой серии мы опишем фундаментальное изменение реализации, сделанное совсем недавно (сентябрь 2013 года) в оригинальной параллельной программе. 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 традиционной многопроцессорной и современной многоядерной программированием.

Как уже упоминалось, "удивительный лабиринт" проект "`amazing`" поставляется совместно с компилятором GNAT Pro. Найдите его в каталоге `share/examples/gnat/amazing`, расположенном под корневой установкой вашего компилятора. Обратите внимание, что изменение дизайна появится в будущих 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…