Gem #93: Программирование высокоэффективных многоядерных приложения - Часть 1

Автор: Pat Rogers, AdaCore

Краткое содержание: В данном Gem описывается реализация в Ada программы оценки эффективности «Chameneos-Redux», которая сравнивает показатели многопотокового приложения на многоядерной платформе. Эта серия публикаций будет нацелена на описание разработки и реализации способов создания высокоэффективных версий программы в Ada.

Давайте начнём…

Chameneos-Redux является частью компьютерного языка Shootout, набор тестов, который сравнивает реализации различных языков программирования в различных видах приложений и платформ. Программа обязана выполнить определённое количество рандеву между мифическими существами "chameneos", где каждое существо представлено отдельной нитью. Каждое рандеву симметрично, в том, что участвующие существа могут быть либо вызывающим, либо вызываемым членом любой данной встречи.

Хотя и одноядерные и многоядерные машины используются в тестах, мы фокусируемся только на многоядерных версиях с нашей реализацией Ada. Результаты многоядерных тестов для всех реализаций доступны здесь: Chameneos-Redux

http://shootout.alioth.debian.org/u32q/benchmark.php?test=chameneosredux&lang=all .

Как вы увидите на этой веб-странице, существует несколько реализаций. (Все реализации предоставляются добровольцами, написанными против общих требований.) Самые быстрые реализации в настоящее время написаны на GNU C, GNU C++, SBCL Lisp и GNAT Ada. После этого другие реализации, также написанные на C, Java, C++, Ada и многих других языках, значительно медленнее. На машине разработки автора реализации C++ и Ada имеют практически одинаковые результаты производительности, с небольшим преимуществом для C++, но на официальной машине тестирования наша версия Ada от AdaCore заметно медленнее, чем версия C++. Расхождение расследуется. Общие результаты иногда также колеблются из-за, по-видимому, неожиданных возмущений в аппаратно-программной платформе используемой для измерений, так что версии с самым высоким рейтингом могут двигаться вверх или вниз немного относительно друг друга.

Причина, по которой другие реализации значительно медленнее, заключается в том, что они не используют тот же фундаментальный или одинаковый дизайн, каждая реализация со своим уникальным дизайном программного кода. Таким образом, наш первый вывод, который можно сделать о работе: дизайн тюнинга кода программы – это главный козырь для повышения производительности. Когда дело доходит до требования повышения производительности, никакое количество настройки не может компенсировать по своей сути медленный дизайн кода программы. Например, более ранняя реализация Ada (предоставленная другим добровольцем) использовала асинхронную инструкцию `select`. В дополнение к трудностям в правильном использовании этого утверждения, семантика его такова, что это по своей сути дорого, и применение его было бы так же дорого в любом языке программирования. Хуже того, асинхронная инструкция `select` была "другой частью" инструкции `selective accept` для рандеву и все естественно в цикле. Фактически программа опрашивалась самым дорогим способом, который только можно себе представить. Хуже всего, что этот код оказался в наихудшем месте – в коде, реализующем основное поведение потоков. Как средство для отображения конструкций Ada дизайн кода программы был впечатляющим, но в качестве демонстрации производительности он не был конкурентоспособным.

Напротив, три самых быстрых проекта (включая текущую реализацию Ada) используют общую переменную для этого наиболее важного аспекта реализации. Доступ к общей переменной осуществляется с помощью специальной инструкции процессора компьютера, которая блокирует шину памяти, а затем атомарно считывает и обновляет значение. В этой общей переменной, путём

упаковки в одно общее слово, содержится как числа завершённых встреч, так и тождество существ (*прим. переводчика:* по видимому подразумевается идентификатор присвоенный существу "chameneos"), ожидающих рандеву. Этот подход с одной инструкцией обеспечивает чрезвычайно быстрый метод обновления состояния программы и передачи данных между потоками.

Машинная инструкция процессора – это инструкция "compare-and-swap" (CAS), доступ к которой можно получить как к "встроенному" коду ассемблера реализуемого компилятором GCC. К нему также можно получить доступ, написав машинный код непосредственно – в коде программы, но подход использования компилятора GCC более удобен. Чтобы импортировать его в программу Ada, объявляется подпрограмма, а затем используется `Pragma Import` для завершения объявления подпрограммы, как обычно, но с соглашением "`intrinsic`", потому что это последовательность инструкций, выдаваемая непосредственно компилятором.

```
function Sync_Val_Compare_And_Swap_32
  (Destination : access Unsigned_32;
   Comparand   : Unsigned_32;
   New_Value   : Unsigned_32)
   return Unsigned_32;

pragma Import (Intrinsic, Sync_Val_Compare_And_Swap_32, "__sync_val_compare_and_swap_4");
```

Объявление для встраивания в код программы – это функция, для того чтобы возвратить предыдущее значение общей переменной после атомарной операции установки нового значения. В частности, встроенная инструкция CAS сравнивает значение `Comparand` с тем, что находится по ссылке `Destination.all`, и если они имеют одинаковое значение, записывает `New_Value` в `Destination.all`. Затем вызывающий функцию код может проверить возвращённое значение, чтобы узнать, действительно ли произошло обновление, а также использовать это значение для других целей. (Существует также версия, которая возвращает логическое значение, указывающее, произошёл ли обмен, вместо возвращения предыдущего значения.)

Другое требование для импорта встраиваемого кода импортированных функций – это способность перегружаться. Существует несколько форматов CAS инструкции в процессоре, в зависимости от размера данных. В результате должна быть выбрана функция для встраивания с соответствующим форматом инструкции CAS . GNAT в настоящее время автоматически не разрешает перегруженные внутренние операции, поэтому параметр `External_Name` для импорта в директиве `pragma` должен определять, какую версию функции необходимо использовать. Для этого к имени добавляется суффикс. В приведённом выше объявлении мы используем 32-битные значения, поэтому мы указываем имя, как показано, в котором суффикс "`_4`" указывает количество байтов для манипулирования и, следовательно, версию встроенной функции, которую желаем использовать.

В будущем Gem мы рассмотрим дополнительные шаги, используемые для повышения производительности, в том числе:

- определяемый пользователем распределитель памяти кучи, который гарантирует, что вся выделенная память из кучи будет выровнена по размеру адресуемой единицы памяти, которая выделяется в кэше процессора. Настраивая это путём указания соблюдения ограничений в языке Ada;
- назначая потоки ядрам, чтобы потоки выполнялись с максимальной производительностью.

Связанный со статьёй текст программы

Attached Files отсутствуют

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

Об авторе

Пэт Роджерс (Pat Rogers) был профессионалом в области вычислительной техники с 1975 года, в основном работая над приложениями на основе микропроцессора в режиме реального времени на языках Ada, C, C ++ и других языках, включая высокопроизводительные имитаторы полета и системы контроля и сбора данных (SCADA), контролирующие опасные материалы. Впервые узнав язык программирования Ada в 1980 году, он был директором лаборатории Ada9X для совместной программы Advanced Strike Technology для ВВС США, исследователя принципов в распределённых системах и исследовательских проектов отказоустойчивости с использованием Ada для ВВС США и армии, а также помощника директора по исследованиям в NASA Software Engineering Research Center. У него есть B.S. и M.S. степени в области проектирования компьютерных систем и компьютерных наук Университета Хьюстона и доктора философии. в информатике из Университета Йорка, Англия. Являясь членом старшего технического персонала AdaCore, он специализируется на поддержке разработчиков в режиме реального времени / встроенных систем, создает и предоставляет учебные курсы, а также является руководителем проекта и разработчиком плагина GNATbench Eclipse для Ada. Он также имеет чёрный пояс 3-го Дан в Тэ Квон До и является основателем клуба AdaCore «The Wicked Uncles».

Last Updated: 10/13/2017
Posted on: 10/25/2010

Обсуждение…