Gem #98: Программирование высокоэффективных многоядерных приложения - Часть 2

Автор: Pat Rogers, AdaCore

Краткое содержание: Данный Gem - второй в серии публикаций, в которых 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 Computer Language Shootout, набора тестов, который сравнивает реализации различных языков программирования в различных приложениях и платформах. Программа должна выполнять определённое количество рандеву между мифическими существами «chameneos», где каждое существо представлено отдельной нитью. Каждое рандеву является симметричным, поскольку участвующие существа могут быть либо вызывающим, либо вызываемым членом в любой заданной встрече. Многоядерные результаты тестов для всех реализаций Chameneos-Redux доступны здесь: Chameneos-Redux

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

В предыдущем Gem в этой серии мы отметили, что дизайн превосходит настройку: когда дело доходит до производительности, никакая настройка не может компенсировать изначально медленный дизайн. В этой статье Gem мы исследуем ещё один важный аспект дизайна: минимизация конфликтов ресурсов с помощью назначений процессоров для потоков. Эта проблема возникает, потому что тест Chameneos-Redux требует двух различных "игр", в которых несколько потоков выполняют необходимое количество рандеву. (Каждая из двух игр имеет разное количество потоков, но обе должны выполнить одинаковое количество рандеву.) Когда доступно несколько процессоров, программа запускает две игры одновременно, поэтому возможно, что два набора потоков захотят выполняться на одном ядре.

Чтобы предотвратить одновременное вмешательство потоков одной игры в потоки другой игры, мы назначаем потоки процессорам на постоянной основе, а не позволяем операционной системе назначать их динамически. В частности, мы разрешаем потоку выполняться на любом из ядер в пределах одного назначенного процессора. Все лидеры тестов Chameneos-redux в своей реализации используют этот же подход, чтобы предотвратить конфликт ресурсов.

Назначение потоков процессорам задаётся в терминах "слотов", а не непосредственно в терминах процессоров. Слот является целым числом, соответствующим номеру процессора, но так как существует, вероятно, больше потоков, чем процессоров, когда номер слота превышает количество процессоров, присутствующих в машине, номер слота "оборачивается" назад к номеру начинающего процессора. В результате фактически отсутствует ограничение на количество доступных слотов. Это не мешает потокам совместно использовать (ядра на) процессоры, это просто делает назначение удобным. Если потоков слишком много, конфликт общего доступа к ресурсам будет неизбежен.

Назначение слотов достигается с помощью директивы `pragma Task_Info` в объявлении типа задачи, представляющей потоки существ chameneos. Аргумент директивы pragma - это значение доступа, обозначающее значение типа `Thread_Attributes`, которое в тестовой операционной системе является типом записи, содержащим один компонент битовой маски сходства. Маска сходства указывает ядра, на которых могут выполняться задачи. Мы определяем сходство функций, которое возвращает значение этого типа. В следующем фрагменте кода показано использование директивы pragma и функции:

```
task type Thread (This : access Creature;  Slot : Natural) is
  pragma Task_Info (new Thread_Attributes'(CPU_Affinity => Affinity (Slot)));
end Thread;
```

Аргументом функции `Affinity` является дискриминант типа задачи, указывающий используемый слот. Логически слот содержит маску соответствия, которая указывает на ядра в соответствующем процессоре, и именно эта маска возвращается функцией.

Нулевой слот используется для хранения битовой маски, указывающей все известные системе ядра. Мы импортируем функцию `Sched_Getaffinity`, чтобы получить это значение. Например, представьте, что функция возвращает маску с включёнными первыми восемью битами, указывающую, что доступно в общей сложности восемь ядер. Нулевой слот будет содержать битовую маску с восемью битами. Предполагая, что два ядра на процессор, и предполагая, что каждые два смежных бита, представляют ядра на том же процессоре, следующее иллюстрирует результирующие маски на слот:

```
            Bit #
Slot #   0123456789...
    0    1111111100
    1    1100000000
    2    0011000000
    3    0000110000
    4    0000001100
    5    1100000000
    6    0011000000
```

Обратите внимание, что в слоте 5 снова задействованы два ядра первого процессора. Вы можете увидеть детали реализации, изучив пакет Chameneos.Processors.

Возвращаясь к игре, основная программа определяет, является ли цель многоядерной машиной и назначает слоты для двух необходимых игр, и, соответственно, потоки в играх:

```
if Processor_Count < 4 then  -- run the games sequentially
   Game1.Start (Game1_Creature_Colors, N, Slot => 0);
   Game1.Await_Completion;
   Game2.Start (Game2_Creature_Colors, N, Slot => 0);
   Game2.Await_Completion;
else -- run the games in parallel
   Game1.Start (Game1_Creature_Colors, N, Slot => 1);
   Game2.Start (Game2_Creature_Colors, N, Slot => 2);
   Game1.Await_Completion;
   Game2.Await_Completion;
end if;
```

Таким образом, две игры не разделяют один и тот же процессор (ядро процессора), поэтому они не конфликтуют друг с другом.

В следующей серии статей мы рассмотрим другой аспект реализации относительно кэша, а именно определяемый пользователем распределитель памяти, который выделяет динамическую память на границах, выровненных по кэшу.

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: 1/31/2011

Обсуждение…