Операционные системы

Управление центральным процессором и объединение ресурсов

Управление центральным процессором…

Реализация многопоточности с использованием технологии OpenMP

Стандарт OpenMP

- Стандарт OpenMP был разработан в 1997г. как API, ориентированный на написание портируемых многопоточных приложений. Сначала он был основан на языке Fortran, но позднее включил в себя и C/C++. Последняя версия OpenMP — 2.0; ее полностью поддерживает Visual C++ 2005.
- http://www.microsoft.com/Rus/Msdn/Magazine/2005/10/OpenMP.mspx

Активизация OpenMP

- Прежде чем заниматься кодом, вы должны знать, как активизировать реализованные в компиляторе средства OpenMP. Для этого служит появившийся в Visual C++ 2005 параметр компилятора /openmp.
- Встретив параметр /openmp, компилятор определяет символ _OPENMP, с помощью которого можно выяснить, включены ли средства OpenMP. Для этого достаточно написать #ifndef _OPENMP.

Параллельная обработка в OpenMP

- Работа OpenMP-приложения начинается с единственного потока (основного). В приложении могут содержаться параллельные регионы, входя в которые, основной поток создает группы потоков (включающие основной поток).
- В конце параллельного региона группы потоков останавливаются, а выполнение основного потока продолжается. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

Конструкции OpenMP

- OpenMP прост в использовании и включает лишь два базовых типа конструкций: директивы *pragma* и функции исполняющей среды OpenMP.
- Директивы pragma, как правило, указывают компилятору реализовать параллельное выполнение блоков кода. Все эти директивы начинаются с *#pragma omp*. Как и любые другие директивы *pragma*, они игнорируются компилятором, не поддерживающим конкретную технологию - в данном случае OpenMP.

Конструкции OpenMP

- Функции OpenMP служат в основном для изменения и получения параметров среды. Кроме того, OpenMP включает API-функции для поддержки некоторых типов синхронизации. Чтобы задействовать эти функции библиотеки OpenMP периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл *omp.h*. Если вы используете в приложении только OpenMP-директивы *pragma*, включать этот файл не требуется.

Формат директивы pragma

- Для реализации параллельного выполнения блоков приложения нужно просто добавить в код директивы pragma и, если нужно, воспользоваться функциями библиотеки OpenMP периода выполнения. Директивы *pragma* имеют следующий формат:

  #pragma omp <директива> [раздел [[,] раздел]...]

Директивы pragma

- OpenMP поддерживает директивы *parallel, for, parallel for, section, sections, single, master, critical, flush, ordered* и *atomic*, которые определяют или механизмы разделения работы или конструкции синхронизации.
- Далее мы рассмотрим простейший пример с использованием директив *parallel, for, parallel for.*

Реализация параллельной обработки

- Самая важная и распространенная директива - *parallel*. Она создает параллельный регион для следующего за ней структурированного блока, например:

  #pragma omp parallel [раздел[[,] раздел]...]
  структурированный блок

Реализация параллельной обработки

- Директива *parallel* сообщает компилятору, что структурированный блок кода должен быть выполнен параллельно, в нескольких потоках.
- Каждый поток будет выполнять один и тот же поток команд, но не один и тот же набор команд — все зависит от операторов, управляющих логикой программы, таких как *if-else*.

Пример параллельной обработки

- В качестве примера рассмотрим классическую программу «Hello World»:

```
#pragma omp parallel
{
 printf("Hello World\n");
}
```

Пример параллельной обработки

- В двухпроцессорной системе вы, конечно же, рассчитывали бы получить следующее:
  Hello World Hello World
- Тем не менее, результат мог быть другим:
  HellHell oo WorWlodrl d
- Второй вариант возможен из-за того, что два выполняемых параллельно потока могут попытаться вывести строку одновременно.

Директива #pragma omp for

- Директива *#pragma omp for* сообщает, что при выполнении цикла *for* в параллельном регионе итерации цикла должны быть распределены между потоками группы.
- Следует отметить, что в конце параллельного региона выполняется барьерная синхронизация (barrier synchronization). Иначе говоря, достигнув конца региона, все потоки блокируются до тех пор, пока последний поток не завершит свою работу.

Пример параллельной обработки

```
#pragma omp parallel
{
 #pragma omp for
 for(int i = 1; i < size; ++i)
      x[i] = (y[i-1] + y[i+1])/2;
}
```

Директива #pragma omp parallel for

- Так как циклы являются самыми распространенными конструкциями, где выполнение кода можно распараллелить, OpenMP поддерживает сокращенный способ записи комбинации директив #pragma omp parallel и #pragma omp for: #pragma omp parallel for.

Пример параллельной обработки

```
#pragma omp parallel for
for(int i = 1; i < size; ++i)
 x[i] = (y[i-1] + y[i+1])/2;
```

Задание числа потоков

- Чтобы узнать или задать число потоков в группе, используйте функции omp_get_num_threads и omp_set_num_threads.
- Первая возвращает число потоков, входящих в текущую группу потоков. Если вызывающий поток выполняется не в параллельном регионе, эта функция возвращает 1.
- Метод omp_set_num_thread задает число потоков для выполнения следующего параллельного региона, который встретится текущему выполняемому потоку (статическое планирование).

Алгоритмы планирования

- По умолчанию в OpenMP для планирования параллельного выполнения циклов *for* применяется алгоритм, называемый статическим планированием. Это означает, что все потоки из группы выполняют одинаковое число итераций цикла.
- Если n - число итераций цикла, а T - число потоков в группе, каждый поток выполнит n/T итераций.

Алгоритмы планирования

- Однако OpenMP поддерживает и другие механизмы планирования, оптимальные в разных ситуациях:
  - динамическое планирование (dynamic scheduling);
  - планирование в период выполнения (runtime scheduling);
  - управляемое планирование (guided scheduling).

Алгоритмы планирования

- Чтобы задать один из этих механизмов планирования, используйте раздел schedule в директиве *#pragma omp for* или *#pragma omp parallel for*.
- Формат этого раздела выглядит так:

  schedule(алгоритм планирования[, число итераций])

Динамическое планирование

- При динамическом планировании каждый поток выполняет указанное число итераций (по умолчанию равно 1).
- После того как поток завершит выполнение заданных итераций, он переходит к следующему набору итераций. Так продолжается, пока не будут пройдены все итерации. Последний набор итераций может быть меньше, чем изначально заданный.

Управляемое планирование

- При управляемом планировании число итераций, выполняемых каждым потоком, определяется по следующей формуле:

  число_выполняемых_потоком_итераций =
  max(число_нераспределенных_итераций/
  omp_get_num_threads(), число итераций)

Примеры задания алгоритмов планирования

```
#pragma omp parallel for
schedule(dynamic, 15)
for(int i = 0; i < 100; ++i) ...

#pragma omp parallel
 #pragma omp for schedule(guided)
```

Планирование в период выполнения

- Планирование в период выполнения - это скорее даже не алгоритм планирования, а способ динамического выбора одного из трех описанных алгоритмов.
- Если в разделе *schedule* указан параметр *runtime*, исполняющая среда OpenMP использует алгоритм планирования, заданный для конкретного цикла for при помощи переменной *OMP_SCHEDULE*.

Планирование в период выполнения

- Переменная *OMP_SCHEDULE* имеет формат «тип[,число итераций]», например:

  set OMP_SCHEDULE=dynamic,8
- Планирование в период выполнения дает определенную гибкость в выборе типа планирования, при этом по умолчанию применяется статическое планирование.