Параллельные заметки №3 — базовые конструкции OpenMP

Начнем знакомство непосредственно с использованием технологии OpenMP и рассмотрим в этой заметке некоторые базовые конструкции.  
  
При использовании OpenMP мы добавляем в программу два вида конструкций: функции исполняющей среды OpenMP и специальные директивы #pragma.

Функции

Функции OpenMP носят скорее вспомогательный характер, так как реализация параллельности осуществляется за счет использования директив. Однако в ряде случаев они весьма полезны и даже необходимы. Функции можно разделить на три категории: функции исполняющей среды, функции блокировки/синхронизации и функции работы с таймерами. Все эти функции имеют имена, начинающиеся с omp\_, и определены в заголовочном файле omp.h. К рассмотрению функций мы вернемся в следующих заметках.

Директивы

Конструкция #pragma в языке Си/Си++ используется для задания дополнительных указаний компилятору. С помощью этих конструкций можно указать как осуществлять выравнивание данных в структурах, запретить выдавать определенные предупреждения и так далее. Форма записи:

#pragma директивы

Использование специальной ключевой директивы «omp» указывает на то, что команды относятся к OpenMP. Таким образом директивы #pragma для работы с OpenMP имеют следующий формат:

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

Как и любые другие директивы pragma, они игнорируются теми компиляторами, которые не поддерживают данную технологию. При этом программа компилируется без ошибок как последовательная. Это особенность позволяет создавать хорошо переносимый код на базе технологии OpenMP. Код содержащий директивы OpenMP может быть скомпилирован Си/Си++ компилятором, который ничего не знает об этой технологии. Код будет выполнятся как последовательный, но это лучше, чем делать две ветки кода или расставлять множество #ifdef.  
OpenMP поддерживает директивы private, parallel, for, section, sections, single, master, critical, flush, ordered и atomic и ряд других, которые определяют механизмы разделения работы или конструкции синхронизации.

Директива parallel

Самой главной можно пожалуй назвать директиву parallel. Она создает параллельный регион для следующего за ней структурированного блока, например:

#pragma omp parallel [другие директивы]

структурированный блок

Директива parallel указывает, что структурный блок кода должен быть выполнен параллельно в несколько[потоков](http://www.viva64.com/terminology/Thread_Parallel_thread_rus.html). Каждый из созданных потоков выполнит одинаковый код содержащийся в блоке, но не одинаковый набор команд. В разных потоках могут выполняться различные ветви или обрабатываться различные данные, что зависит от таких операторов как if-else или использования директив распределения работы.  
  
Чтобы продемонстрировать запуск нескольких потоков, распечатаем в распараллеливаемом блоке текст:

#pragma omp parallel

{

cout << "OpenMP Test" << endl;

}

На 4-х ядерной машине мы можем ожидать увидеть следующей вывод

OpenMP Test

OpenMP Test

OpenMP Test

OpenMP Test

Но на практике я получил следующий вывод:

OpenMP TestOpenMP Test

OpenMP Test

OpenMP Test

Это объясняется совместным использованием одного ресурса из нескольких потоков. В данном случае мы выводим на одну консоль текст в четырех потоках, которые никак не договариваются между собой о последовательности вывода. Здесь мы наблюдаем возникновение [состояния гонки](http://www.viva64.com/terminology/Race_condition_rus.html) (race condition).  
  
Состояние гонки — ошибка проектирования или реализации многозадачной системы, при которой работа системы зависит от того, в каком порядке выполняются части кода. Эта разновидность ошибки являются наиболее распространенной при параллельном программировании и весьма коварна. Воспроизведение и локализация этой ошибки часто бывает затруднена в силу непостоянства своего проявления (смотри также термин [Гейзенбаг](http://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%B9%D0%B7%D0%B5%D0%BD%D0%B1%D0%B0%D0%B3)).

Директива for

Рассмотренный нами выше пример демонстрирует наличие параллельности, но сам по себе он бессмыслен. Теперь извлечем пользу из параллельности. Пусть нам необходимо извлечь корень из каждого элемента массива и поместить результат в другой массив:

void VSqrt(double \*src, double \*dst, ptrdiff\_t n)

{

for (ptrdiff\_t i = 0; i < n; i++)

dst[i] = sqrt(src[i]);

}

Если мы напишем:

#pragma omp parallel

{

for (ptrdiff\_t i = 0; i < n; i++)

dst[i] = sqrt(src[i]);

}

то мы вместо ускорения впустую проделаем массу лишней работы. Мы извлечем корень из всех элементов массива в каждом потоке. Для того, чтобы распараллелить цикл нам необходимо использовать директиву разделения работы «for». Директива #pragma omp for сообщает, что при выполнении цикла for в параллельном регионе итерации цикла должны быть распределены между потоками группы:

#pragma omp parallel

{

#pragma omp for

for (ptrdiff\_t i = 0; i < n; i++)

dst[i] = sqrt(src[i]);

}

Теперь каждый создаваемый поток будет обрабатывать только отданную ему часть массива. Например, если у нас 8000 элементов, то на машине с четырьмя ядрами работа может быть распределена следующим образом. В первом потоке переменная i принимает значения от 0 до 1999. Во втором от 2000 до 3999. В третьем от 4000 до 5999. В четвертом от 6000 до 7999. Теоретически мы получаем ускорение в 4 раза. На практике ускорение будет чуть меньше из-за необходимости создать потоки и дождаться их завершения. В конце параллельного региона выполняется барьерная синхронизация. Иначе говоря, достигнув конца региона, все потоки блокируются до тех пор, пока последний поток не завершит свою работу.  
  
Можно использовать сокращенную запись, комбинируя несколько директив в одну управляющую строку. Приведенный выше код будет эквивалентен:

#pragma omp parallel for

for (ptrdiff\_t i = 0; i < n; i++)

dst[i] = sqrt(src[i]);

Директивы private и shared

Относительно параллельных регионов данные могут быть общими (shared) или частными (private). Частные данные принадлежат потоку и могут быть модифицированы только им. Общие данные доступны всем потокам. В рассматриваемом ранее примере массив представлял общие данные. Если переменная объявлена вне параллельного региона, то по умолчанию она считается общей, а если внутри то частной. Предположим, что для вычисления квадратного корня нам необходимо использовать промежуточную переменную value:

double value;

#pragma omp parallel for

for (ptrdiff\_t i = 0; i < n; i++)

{

value = sqrt(src[i]);

dst[i] = value;

}

В приведенном коде переменная value объявлена вне параллельного региона, задаваемого директивами "#pragma omp parallel for", а значит является общей (shared). В результате переменная value начнет использоваться всеми потоками одновременно, что приведет к ошибке состояния гонки и на выходе мы получим мусор.  
  
Чтобы сделать переменную для каждого потока частной (private) мы можем использовать два способа. Первый — объявить переменную внутри параллельного региона:

#pragma omp parallel for

for (ptrdiff\_t i = 0; i < n; i++)

{

double value;

value = sqrt(src[i]);

dst[i] = value;

}

Второй — воспользоваться директивой private. Теперь каждый поток будет работать со своей переменной value:

double value;

#pragma omp parallel for private(value)

for (ptrdiff\_t i = 0; i < n; i++)

{

value = sqrt(src[i]);

dst[i] = value;

}

Помимо директивы private, существует директива shared. Но эту директиву обычно не используют, так как и без нее все переменные объявленные вне параллельного региона будут общими. Директиву можно использовать для повышения наглядности кода.  
  
Мы рассмотрели только малую часть директив OpenMP и продолжим знакомство с ними в следующих уроках.

* [openmp](http://habrahabr.ru/tag/openmp/)
* , [parallel programming](http://habrahabr.ru/tag/parallel%20programming/)
* , [параллельное программирование](http://habrahabr.ru/tag/%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/)

**+21**

23 февраля 2010, 09:17

18

[**Andrey2008**](http://habrahabr.ru/users/Andrey2008/)

комментарии (6)

* + [spiff](http://habrahabr.ru/users/spiff/)
  + [**spiff**](http://habrahabr.ru/users/spiff/),
  + 24 февраля 2010, 07:36
  + [#](http://habrahabr.ru/company/intel/blog/85273/#comment_2548261)
    - **+1**

Хороший, краткий, доступный обзор. Буду ждать продолжения.

* + [sergehog](http://habrahabr.ru/users/sergehog/)
  + [**sergehog**](http://habrahabr.ru/users/sergehog/),
  + 24 февраля 2010, 10:41
  + [#](http://habrahabr.ru/company/intel/blog/85273/#comment_2548516)
    - **+1**

Супер!   
Каждый раз когда писал громоздкие «for» мечтал о подобной параллелизации. В ближайшее же время опробую.  
  
Вот еще какой вопрос знатокам, сложно ли это прикрутить это к gcc или другим компиляторам кроме Visual Studio?

* + - [spybot](http://habrahabr.ru/users/spybot/)
    - [**spybot**](http://habrahabr.ru/users/spybot/),
    - 24 февраля 2010, 10:53
    - [#](http://habrahabr.ru/company/intel/blog/85273/#comment_2548539)
    - [↑](http://habrahabr.ru/company/intel/blog/85273/#comment_2548516)
      * **+1**

GCC поддерживает OpenMP с версии 4.2.

* + - [reddot](http://habrahabr.ru/users/reddot/)
    - [**reddot**](http://habrahabr.ru/users/reddot/),
    - 26 февраля 2010, 12:32
    - [#](http://habrahabr.ru/company/intel/blog/85273/#comment_2557011)
    - [↑](http://habrahabr.ru/company/intel/blog/85273/#comment_2548516)
      * **+1**

reddot@doone:~/default/samples/openmp$ sudo apt-get install libgomp1  
reddot@doone:~/default/samples/openmp$ cat hello.c  
#include <stdio.h>  
  
int main()  
{  
#pragma omp parallel  
printf(«hello world\n»);  
return 0;  
}  
  
reddot@doone:~/default/samples/openmp$ gcc -Wall -Wextra -fopenmp hello.c  
reddot@doone:~/default/samples/openmp$ ./a.out  
hello world  
hello world

* + [Unforgiven](http://habrahabr.ru/users/Unforgiven/)
  + [**Unforgiven**](http://habrahabr.ru/users/Unforgiven/),
  + 24 февраля 2010, 11:31
  + [#](http://habrahabr.ru/company/intel/blog/85273/#comment_2548650)
    - **+1**

Есть ли средства отладки (оч желательно — бесплатные), чтобы увидеть сколько потоков запущены, какие переменные в каждом из них, над какими данными выполняются операции и т д...?

* + - [izard](http://habrahabr.ru/users/izard/)
    - [**izard**](http://habrahabr.ru/users/izard/),
    - 24 февраля 2010, 16:42
    - [#](http://habrahabr.ru/company/intel/blog/85273/#comment_2550055)
    - [↑](http://habrahabr.ru/company/intel/blog/85273/#comment_2548650)
      * **0**

Intel Thread profiler — показывает достаточно наглядно, но это не совсем отладчик. Бесплатный первый месяц, потом можно повторить.