Лабораторная работа №1.

Компиляция программы. Модель параллельной программы. Директивы и функции. Выполнение программы. Замер времени.

Компиляция программы

Для использования механизмов OpenMP нужно скомпилировать программу компилятором, поддерживающим OpenMP, с указанием соответствующего ключа (например, в icc/ifort используется ключ компилятора **`-openmp`**, в gcc /gfortran **`-fopenmp`**, Sun Studio **`-xopenmp`**, в Visual C++ **`- /openmp`**, в PGI **`-mp`**). Компилятор интерпретирует директивы OpenMP и создаёт параллельный код. При использовании компиляторов, не поддерживающих OpenMP, директивы OpenMP игнорируются без дополнительных сообщений. Компилятор с поддержкой OpenMP определяет макрос **`_OPENMP`**, который может использоваться для условной компиляции отдельных блоков, характерных для параллельной версии программы. Этот макрос определён в формате **`yyyymm`**, где **`yyyy`** и **`mm`** –цифры года и месяца, когда был принят поддерживаемый стандарт OpenMP. Например, компилятор, поддерживающий стандарт OpenMP 3.0, определяет **`OPENMP`** в **`200805`**.

Для проверки того, что компилятор поддерживает какую-либо версию OpenMP, достаточно написать директивы условной компиляции **`#ifdef`** или **`#ifndef`**. Простейший пример условной компиляции в программе на языке Си приведен в примере 1.

```
#include <stdio.h>
int main(){
#ifdef _OPENMP
printf("OpenMP is supported!\n");
#endif
}
```

Пример *1*. Условная компиляция на языке Си.

Модель параллельной программы

Распараллеливание в OpenMP выполняется явно при помощи вставки в текст программы специальных директив, а также вызова вспомогательных функций. При использовании OpenMP предполагается *SPMD*-модель *(Single Program Multiple Data)* параллельного программирования, в рамках которой для всех параллельных нитей используется один и тот же код.

Программа начинается с последовательной области – сначала работает один процесс (нить), при входе в параллельную область порождается ещё некоторое число процессов, между которыми в дальнейшем распределяются части кода. По завершении параллельной области все нити, кроме одной (нити-мастера), завершаются, и начинается последовательная область. В программе может быть любое количество параллельных и последовательных областей. Кроме того, параллельные области могут быть также вложенными друг в

друга. В отличие от полноценных процессов, порождение нитей является относительно быстрой операцией, поэтому частые порождения и завершения нитей не так сильно влияют на время выполнения программы.

Для написания эффективной параллельной программы необходимо, чтобы все нити, участвующие в обработке программы, были равномерно загружены полезной работой. Это достигается тщательной балансировкой загрузки, для чего предназначены различные механизмы OpenMP.

Существенным моментом является также необходимость синхронизации доступа к общим данным. Само наличие данных, общих для нескольких нитей, приводит к конфликтам при одновременном несогласованном доступе. Поэтому значительная часть функциональности OpenMP предназначена для осуществления различного рода синхронизаций работающих нитей.

OpenMP не выполняет синхронизацию доступа различных нитей к одним и тем же файлам. Если это необходимо для корректности программы, пользователь должен явно использовать директивы синхронизации или соответствующие библиотечные функции. При доступе каждой нити к своему файлу никакая синхронизация не требуется.

Директивы и функции

Значительная часть функциональности OpenMP реализуется при помощи директив компилятору. Они должны быть явно вставлены пользователем, что позволит выполнять программу в параллельном режиме. Директивы OpenMP в программах на языке Си оформляются указаниями препроцессору, начинающимися с **`#pragma omp`**. Ключевое слово **`omp`** используется для того, чтобы исключить случайные совпадения имён директив OpenMP с другими именами.

Формат директивы на Си/Си++:

`#pragma omp` ***`directive-name`*** **[**опция **[,]** опция**...]**

Объектом действия большинства директив является один оператор или блок, перед которым расположена директива в исходном тексте программы. В OpenMP такие операторы или блоки называются ассоциированными с директивой. Ассоциированный блок должен иметь одну точку входа в начале и одну точку выхода в конце. Порядок опций в описании директивы несущественен, в одной директиве большинство опций может встречаться несколько раз. После некоторых опций может следовать список переменных, разделяемых запятыми.

Все директивы OpenMP можно разделить на 3 категории: определение параллельной области, распределение работы, синхронизация. Каждая директива может иметь несколько дополнительных атрибутов – опций (*clause*). Отдельно специфицируются опции для назначения классов переменных, которые могут быть атрибутами различных директив.

Чтобы задействовать функции библиотеки *OpenMP* периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл **`omp.h`**

Если вы используете в приложении только OpenMP-директивы, включать этот файл не требуется. Функции назначения параметров имеют приоритет над соответствующими переменными окружения.

Все функции, используемые в OpenMP, начинаются с префикса **omp_**. Если пользователь не будет использовать в программе имён, начинающихся с такого префикса, то конфликтов с объектами OpenMP заведомо не будет. В языке Си, кроме того, является существенным регистр символов в названиях функций. Названия функций OpenMP записываются строчными буквами. Для того чтобы программа, использующая функции OpenMP, могла оставаться корректной для обычного компилятора, можно прилинковать специальную библиотеку, которая определит для каждой функции соответствующую «заглушку» (*stub*). Например, в компиляторе Intel соответствующая библиотека подключается заданием ключа **–openmp-stubs**.

Выполнение программы

После получения выполняемого файла необходимо запустить его на требуемом количестве процессоров. Для этого обычно нужно задать количество нитей, выполняющих параллельные области программы, определив значение переменной среды **OMP_NUM_THREADS**. Например, в Linux в командной оболочке bash это можно сделать при помощи следующей команды:

```
export OMP_NUM_THREADS=n
```

После запуска начинает работать одна нить, а внутри параллельных областей одна и та же программа будет выполняться всем набором нитей. Стандартный вывод программы в зависимости от системы будет выдаваться на терминал или записываться в файл с предопределенным именем.

Замер времени

В OpenMP предусмотрены функции для работы с системным таймером. Функция **omp_get_wtime()** возвращает в вызвавшей нити астрономическое время в секундах (вещественное число двойной точности), прошедшее с некоторого момента в прошлом.

```
double omp_get_wtime(void)
```

Если некоторый участок программы окружить вызовами данной функции, то разность возвращаемых значений покажет время работы данного участка.

Гарантируется, что момент времени, используемый в качестве точки отсчета, не будет изменён за время существования процесса. Таймеры разных нитей могут быть не синхронизированы и выдавать различные значения.

Функция **omp_get_wtick()** возвращает в вызвавшей нити разрешение таймера в секундах. Это время можно рассматривать как меру точности таймера.

```
double omp_get_wtick(void)
```

Пример 2 иллюстрирует применение функций **omp_get_wtime()** и **omp_get_wtick()** для работы с таймерами в OpenMP. В данном примере производится замер начального времени, затем сразу замер конечного времени.

Разность времён даёт время на замер времени. Кроме того, измеряется точность системного таймера.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
double start_time, end_time, tick;
start_time = omp_get_wtime();
end_time = omp_get_wtime();
tick = omp_get_wtick();
printf("Время на замер времени %lf\n", end_time-start_time);
printf("Точность таймера %lf\n", tick);
}
```

Пример *2* Работа с системными таймерами на языке Си.

Задания

1. Определите, какую версию стандарта OpenMP поддерживает компилятор на доступной системе.
2. Откомпилируйте любую последовательную программу с включением опций поддержки технологии OpenMP и запустите с использованием нескольких нитей. Сколько нитей будет реально исполнять операторы данной программы?
3. Может ли программа на OpenMP состоять только из параллельных областей? Только из последовательных областей?
4. Чем отличается нить-мастер от всех остальных нитей?
5. При помощи функций OpenMP попробуйте определить время, необходимое для работы функции **`omp_get_wtick()`**. Хватает ли для этого точности системного таймера?