МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ДГТУ)

Кафедра "Программное обеспечение вычислительной техники и автоматизированных систем"

Технология OpenMP

Методические указания к практическим работам по дисциплине "Параллельные вычисления"

Ростов-на-Дону

20 г.

Составитель: к.ф.-м.н., доц. Габрельян Б.В.

УДК 512.3

Технология OpenMP: методические указания – Ростов н/Д: Издательский центр ДГТУ, 20 . – с.

В методической разработке рассматриваются вопросы поддержки параллельного программирования в технологии OpenMP для языка C++. Даны задания по выполнению лабораторной работы. Методические указания предназначены для магистрантов направления 01.05.00 – «Математическое обеспечение и администрирование информационных систем».

Цель работы: Ознакомление с технологией OpenMP для C++.

I. Директивы OpenMP.

Стандарт OpenMP ориентирован на языки программирования Си/C++ и Fortran. В настоящем пособии рассматривается только то, что относится к языкам Си/C++. В MS Visual Studio чтобы использовать OpenMP необходимо в свойствах проекта в разделе Configuration Properties выбрать C/C++, затем Language и задать в Open MP Support **Yes (/openpm)**.

В Си/C++ директивы OpenMP имеют вид прагм, обрабатываемых препроцессором. Их общий синтаксис таков:

#pragma omp директива [предложение [предложение] …]

Обычно директивы применяются к части программы, последовательному набору операторов, т.е. к некоторому структурному блоку. Структурный блок задается как обычный блок, т.е. с помощью фигурных скобок. Например,

```
#pragma omp parallel
{
    // параллельный участок программы
}
```

определяет распараллеленный блок кода, который будет выполняться в N потоках (нитях).

Количество нитей может задаваться в программе или определяется значением переменной окружения процесса OMP_NUM_THREADS. У каждой нити будет свой номер в диапазоне от 0 до OMP_NUM_THREADS-1.

Предложения shared, private, default определяют режим использования указанных в них переменных в структурном блоке. shared задает переменные, общие для всех потоков, private – локальные переменные для каждого потока, default объявляет все переменные, заданные в блоке либо как shared, либо как private, либо как none.

```
int x, y, i;
#pragma omp parallel shared(x ,y) private(i)
```

```
{
…
}
```

Предложение firstprivate объявляет локальные переменные, инициализируемые в последовательном блоке до входа в параллельный блок программы, lastprivate – переменные, значения которых доступны в последовательной части программы после выхода из параллельного блока.

if(выражение) – если значение выражения рано нулю, то в следующем блоке распараллеливание не проводится. Например,

```
…
int n = 1000;
#pragma omp parallel if( n > 1000 )
{
        // распараллеленная секция
}
```

Распараллеленная секция будет на самом деле выполняться в одном потоке (последовательно) до тех пор, пока значение переменной n меньше 1001.

reduction позволяет выполнить указанную операцию (или вызвать указанную функцию) над одноименными локальными значениями, полученными в разных потоках, с тем, чтобы полученное значение можно было использовать далее в последовательной части программы. Общий синтаксис:

reduction(знак операции или имя функции : список имен переменных)

Разрешены следующие операции: +, -, *, &, |, ^, &&, ||. Например, если количество используемых для распараллеливания потоков numThreads

```
int sum = 0;
#pragma omp parallel reduction(+: sum)
{
        for(int i=0; i<size; i += numThreads) {
                int index = i + omp_get_thread_num();
                sum += a[index] + b[index];
```

```
        }
        #pragma barrier
}

cout << "sum=" << sum << endl;
```

Здесь функция omp_get_thread_num() возвращает номер текущего потока.

#pragma barrier устанавливает барьер, заставляющий основной поток ожидать завершения всех параллельных потоков.

for позволяет организовать цикл, итерации которого будут выполняться параллельно, разделяя доступные потоки. Например,

```
cons tint size = 160;

int a[size], b[size], c[size] = {};

…

#pragma omp parallel for
for(int i=0; i<size; ++i) c[i] = a[i] + b[i];
```

По умолчанию в конце цикла ставится барьер. Если он не нужен, то его нужно явно убрать с помощью предложения nowait в директиве for.

Для синхронизации потоков помимо директивы #pragma omp barrier, можно использовать также директивы atomic и critical.

#pragma omp atomic указывается перед выражением с присваиванием. Изменение переменной в левой части присваивания будет атомарной операцией.

```
int x = 0;

#pragma omp parallel shared(x)

{

…

#pragma omp atomic

        x = x + 1;

…

}
```

#pragma omp critical задает критическую секцию в параллельном участке программы, содержащую код, который должен быть выполнен всеми потоками, но не параллельно. Пока поток, первым попавший в критическую секцию выполняет ее, остальные потоки в которых выполнение дошло до этой секции ждут его завершения. Затем какой-то другой поток монопольно выполняет эту секцию и т.д.

II. Встроенные функции OpenMP.

Прототипы функций OpenMP находятся в файле заголовков omp.h.

Функция int omp_get_num_threads() возвращает количество потоков выполняющихся в данной параллельной секции. В последовательной части программы возвращает 1.

void omp_set_num_threads(int num) задает количество потоков для параллельной секции кода, в которой не используется предложение num_threads.

int omp_get_max_threads() возвращает наибольшее количество потоков, которые могут быть созданы в параллельной секции программы.

int omp_get_thread_num() возвращает номер текущего потока. Главный поток имеет номер 0.

int omp_get_num_procs() возвращает количество процессоров, доступных в момент вызова функции.

int omp_in_parallel()возвращает ноль вне параллельной секции кода.

void omp_set_dynamic(int num) разрешает или запрещает динамическую настройку числа используемых в параллельном блоке потоков. Если num == 0 – запрещает.

int omp_get_dynamic() – если возвращает не ноль, динамическая настройка числа потоков разрешена.

void omp_set_nested(int nested) разрешает или запрещает использование вложенных параллельных блоков. Если nested 0, то вложенные параллельные блоки выполняются последовательно в текущем потоке.

int omp_get_nested() возвращает ноль, если вложенные параллельные блоки запрещены (режим по умолчанию).

double omp_get_wtime() возвращает количество секунд, прошедших с некоторого момента, который выбирается произвольно, но не меняется во время выполнения программы.

double omp_get_wtick() возвращает количество секунд между сигналами от таймера.

III. Задания.

1. Создайте последовательную и параллельную версии какого-нибудь алгоритма численного интегрирования для вычисления значения числа π. Параллельная версия должна использовать возможности OpenMP. Сравните время работы параллельного и последовательного алгоритмов при заданной точности вычисления не меньшей 0.0001.

2. Создайте последовательную и параллельную версии алгоритма сортировки простым выбором. Параллельная версия должна использовать возможности OpenMP и ей в качестве аргумента должно передаваться число используемых потоков. Сравните время работы параллельного и последовательного алгоритмов для массивов разных размеров. Сравните время работы параллельного алгоритма в зависимости от числа используемых потоков.

3. Создайте последовательную и параллельную версии операции умножения для класса Matrix из лабораторной работы "Использование потоков выполнения в Си/C++". Параллельная версия должна использовать возможности OpenMP. Сравните время работы параллельного и последовательного алгоритмов для массивов разных размеров.

IV. Контрольные вопросы.

1. Какими преимуществами по сравнению с другими технологиями распараллеливания обладает технология OpenMP?
2. Каковы ограничения технологии OpenMP?
3. Какая директива распараллеливает тело цикла?
4. Как узнать количество процессоров?

5. Как узнать, сколько потоков выполняются в параллельной секции кода программы?
6. Как узнать номер текущего потока?
7. Как вычислить время, затраченное на выполнение некоторого участка программы?

Литература

1. Левин М.П. "Параллельное программирование с использованием OpenMP: учебное пособие" – М.: Интернет Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2008. – 118 с.

2. "OpenMP C and C++ application program interface" – OpenMP architecture review board, 1997-2002. – 106 p.

3. Chapman B., Jost G., van der Past R. "Using OpenMP. Portable shared memory parallel programming" – London: MIT Press, 2008. – 378 p.