**Лабораторная работа №6**

**Замки и барьеры**

**Цель:** изучить основные особенности использования барьеров и замков в OpenMP на примере использования в рамках языка С++.

Для получения **теоретических сведений** настоятельно рекомендуется при домашней подготовке изучить материалы, представленные в списке литературы в конце разработки, а также прочие материалы по тематике лабораторной работы, представленные в открытых источниках.

Далее следует краткий конспект материала, приведенного в данных источниках, в конце включающий короткие примеры фрагментов программ.

###### 1. Применение переменных семафорного типа (замков)

В OpenMP поддерживается специальный тип данных omp\_lock\_t, который близок к классическому понятию семафоров. Для переменных этого типа определены функции библиотеки OpenMPВ:

- Инициализировать замок:

void omp\_init\_lock(omp\_lock\_t \*lock);

- Установить замок:

void omp\_set\_lock (omp\_lock\_t &lock);

Если при установке замок был установлен ранее, то поток блокируется.

- Освободить замок:

void omp\_unset\_lock (omp\_lock\_t &lock);

После освобождения замка при наличии блокированных на этом замке потоков один из них активизируется и замок снова отмечается как закрытый.

- Установить замок без блокировки:

int omp\_test\_lock (omp\_lock\_t &lock);

Если замок свободен, функция его закрывает и возвращает значение true. Если замок занят, поток не блокируется, и функция возвращает значение false.

- Перевод замка в неинициализированное состояние:

void omp\_destroy\_lock(omp\_lock\_t &lock)

В OpenMP поддерживаются также и вложенные замки, которые предназначены для использования в ситуациях, когда внутри критических секций осуществляется вызов одних и тех же замков. Механизм работы с вложенными замками является тем же самым (в наименование функций добавляется поле nest), т. е. их использование обеспечивается при помощи функций:

void omp\_init\_nest\_lock(omp\_nest\_lock\_t \*lock);

void omp\_set\_nest\_lock (omp\_nest\_lock\_t &lock);

void omp\_unset\_nest\_lock (omp\_nest\_lock\_t &lock);

int omp\_test\_nest\_lock (omp\_nest\_lock\_t &lock);

void omp\_destroy\_nest\_lock(omp\_nest\_lock\_t &lock)

##### 2. Определение однопотоковых участков для параллельных фрагментов (директивы single и master)

При выполнении параллельных фрагментов может оказаться необходимым реализовать часть программного кода только одним потоком (например, открытие файла). Данную возможность в OpenMP обеспечивают директивы single и master.

Формат директивы single имеет вид:

#pragma omp single <параметр> ...] <блок\_программы>

Директива single определяет блок параллельного фрагмента, который должен быть выполнен только одним потоком; все остальные потоки ожидают завершения выполнения данного блока (если не указан параметр nowait).

В качестве параметров директивы могут использоваться:

- private (list)

- firstprivate (list)

- copyprivate (list)

- nowait

Параметр copyprivate обеспечивает копирование переменных, указанных в списке list, после выполнения блока директивы single в локальные переменные всех остальных потоков.

Формат директивы master имеет вид:

#pragma omp master <блок\_программы>

Директива master определяет фрагмент кода, который должен быть выполнен только основным потоком; все остальные потоки пропускают данный фрагмент кода (завершение директивы по умолчанию не синхронизируется).

##### 3. Выполнение барьерной синхронизации (директива barrier)

При помощи директивы barrier можно определить точку синхронизации, которую должны достигнуть все потоки для продолжения вычислений (директива может находиться в пределах как параллельного фрагмента, так и параллельной области, т. е. директива является отделяемой). Формат директивы barrier имеет вид:

#pragma omp barrier

##### 4. Синхронизация состояния памяти (директива flush)

Директива flush позволяет определить точку синхронизации, в которой системой должно быть обеспечено единое для всех потоков состояние памяти (т. е. если потоком какое-либо значение извлекалось из памяти для модификации, то измененное значение обязательно должно быть записано в общую память). Формат директивы flush имеет вид:

#pragma omp flush [(list)]

Следует отметить, что директива flush неявным образом присутствует в директивах barrier, critical, ordered, parallel, for, sections, single.

##### 5. Определение постоянных локальных переменных потоков (директива threadprivate и параметр copyin директивы parallel)

Для потоков могут быть определены локальные переменные (при помощи параметров private, firstprivate, lastprivate), которые создаются в начале соответствующего параллельного фрагмента и удаляются при завершении потоков. В OpenMP имеется возможность создания и постоянно существующих локальных переменных для потоков при помощи директивы threadprivate.

Формат директивы threadprivate имеет вид:

#pragma omp threadprivate (list)

Список list содержит набор определяемых переменных. Созданные локальные копии не видимы в последовательных участках выполнения программы (т. е. вне параллельных фрагментов), но существуют в течение всего времени выполнения программы. Указываемые в списке переменные должны быть уже определены в программе; объявление переменных в директиве должно предшествовать использованию переменных в потоках.

##### 

##### 6. Управление количеством потоков

По умолчанию количество создаваемых потоков определяется реализацией и обычно совпадает с числом имеющихся вычислительных элементов в системе (процессоров и/или ядер) . При необходимости количество создаваемых потоков может быть изменено – для этой цели в OpenMP предусмотрено несколько способов действий.

Прежде всего, количество создаваемых потоков может быть задано при помощи параметра num\_threads директивы parallel.

Количество необходимых потоков может быть также задано при помощи функции omp\_set\_num\_threads библиотеки OpenMP.

Формат функции omp\_set\_num\_threads имеет вид:

void omp\_set\_num\_threads (int num\_threads);

Вызов функции omp\_set\_num\_threads должен осуществляться из последовательной части программы.

Для задания необходимого количества потоков можно воспользоваться и переменной окружения OMP\_NUM\_THREADS. При использовании нескольких способов задания наибольший приоритет имеет параметр num\_threads директивы parallel, затем функция библиотеки, затем переменная окружения.

Изменение количества создаваемых потоков может быть полезно и для целей отладки разрабатываемой параллельной программы с целью проверки ее работоспособности при разном количестве потоков.

Приведем здесь также дополнительные функции библиотеки OpenMP, которые могут быть использованы при работе с потоками:

- Получение максимально-возможного количества потоков:

int omp\_get\_max\_threads(void)

- Получение фактического количества потоков в параллельной области программы:

int omp\_get\_num\_threads(void)

- Получение номера потока:

int omp\_get\_thread\_num(void)

- Получение числа вычислительных элементов (процессоров или ядер), доступных приложению:

int omp\_get\_num\_procs(void)

##### 7. Задание динамического режима при создании потоков

Количество создаваемых потоков в параллельных фрагментах программы по умолчанию является фиксированным (см. также предыдущий пункт), однако стандартом предусматривается возможность динамического режима, когда количество потоков может определяться реализацией для оптимизации функционирования вычислительной системы. Разрешение динамического режима и его отключение осуществляется при помощи функции omp\_set\_dynamic библиотеки OpenMP.

Формат функции omp\_set\_ dynamic имеет вид:

void omp\_set\_dynamic (int dynamic);

Вызов функции omp\_set\_dynamic должен осуществляться из последовательной части программы. Функция разрешает (dynamic=true) или отключает (dynamic=false) динамический режим создания потоков.

Для управления динамическим режимом создания потоков можно воспользоваться и переменной окружения OMP\_DYNAMIC. При использовании нескольких способов задания наибольший приоритет имеет функция библиотеки, затем переменная окружения.

Для получения состояния динамического режима можно воспользоваться функций omp\_get\_dynamic библиотеки OpenMP:

int omp\_get\_dynamic (void);

###### 8. Управление вложенностью параллельных фрагментов

Параллельные фрагменты программы могут быть вложенными – такая возможность определяется реализацией OpenMP. По умолчанию для выполнения вложенных параллельных фрагментов создается столько же потоков, как и для параллельных фрагментов верхнего уровня.

Управление режимом выполнения вложенных фрагментов осуществляется при помощи функции omp\_set\_nested библиотеки OpenMP.

Формат функции omp\_set\_nested имеет вид:

void omp\_set\_nested (int nested);

Вызов функции omp\_set\_nested должен осуществляться из последовательной части программы. Функция разрешает (nested=true) или отключает (nested=false) режим поддержки вложенных параллельных фрагментов.

Для управления режимом поддержки вложенных параллельных фрагментов можно воспользоваться и переменной окружения OMP\_NESTED. При использовании нескольких способов задания наибольший приоритет имеет функция библиотеки, затем переменная окружения.

Для получения состояния режима поддержки вложенных параллельных фрагментов можно воспользоваться функций omp\_get\_nested библиотеки OpenMP:

int omp\_get\_nested (void);

**9. Решение проблемы поддержки единственности программного кода**

Проблемы поддержки единственности программного кода могут быть решены с помощью директив условной компиляции – средств препроцессирования. Стандартом OpenMP предусматривается, что компиляторы с поддержкой технологии OpenMP должны определять макропеременную \_OPENMP, значением которой является дата поддерживаемого стандарта в формате ГГГГММ (год, месяц). Для обеспечения единственности программного кода последовательного и параллельного вариантов программ, все OpenMP-зависимые расширения параллельного варианта программы следует окружать директивами условной компиляции в виде:

#ifdef \_OPENMP

<OpenMP-зависимый программный код>

#endif

Последовательный вариант программы может порождаться и компиляторами с поддержкой OpenMP – для обеспечения такой возможности в составе OpenMP имеются функции-заглушки для всех функций библиотеки OpenMP. Указание подобного режима компиляции осуществляется при помощи соответствующих параметров компилятора.

Пример организации взаимоисключения при помощи замков

**omp\_lock\_t lock;**

**omp\_init\_lock(&lock);**

smax = 0;

#pragma omp parallel shared(smax,a) private(i,j,sum)

{

#pragma omp for

for (i=0; i < NMAX; i++)

{

sum = 0;

for (j=0; j < NMAX; j++)

sum += a[i][j];

printf ("Сумма элементов строки %d равна %f\n",i,sum);

if ( sum > smax )

{

**omp\_set\_lock (&lock);**

if ( sum > smax )

smax = sum;

**omp\_unset\_lock (&lock);**

}

}

} /\* Завершение параллельного фрагмента \*/

printf ("Максимальная сумма равна %f\n",smax);

**omp\_destroy\_lock (&lock);**

**Лабораторные задания**

**Задание 1.** Модифицировать программы, составленные в Л.Р. №№3,4, используя переменные семафорного типа. Измерять время работы программы для тех же значений параметров, что были использованы при выполнении Л.Р. №3. **Результаты сравнить и занести в отчёт.**

**Задание 2.** Модифицировать программы, составленные в Л.Р. №№3,4, используя методы барьерной синхронизации. Измерять время работы программы для тех же значений параметров, что были использованы при выполнении Л.Р. №3. **Результаты сравнить и занести в отчёт.**

**Требования к сдаче работы**

1. При домашней подготовке изучить теоретический материал по тематике лабораторной работы, представленный в списке литературы ниже, выполнить представленные примеры, занести в отчёт результаты выполнения.
2. Продемонстрировать выполнение лабораторных заданий.
3. Ответить на контрольные вопросы.
4. Показать преподавателю отчет.

**Литература**

1. Спецификации стандарта OpenMP (на английском языке):

<http://openmp.org/wp/openmp-specifications/>

2. Материалы, представленные на сайте intuit.ru в рамках курса «Intel Parallel Programming Professional (Introduction)»:

<http://old.intuit.ru/department/supercomputing/ppinteltt/4/>

3. С.А. Лупин, М.А. Посыпкин Технологии параллельного программирования. – М.: ИД «ФОРУМ»: ИНФРА-М, 2011. – С. 119-145. *(Глава, посвященная OpenMP)*

4. Канг Су Гэтлин, Пит Айсенси - OpenMP и C++:

<http://msdn.microsoft.com/ru-ru/library/dd335940.aspx>

5. API OpenMP C и C++

<http://msdn.microsoft.com/ru-ru/library/8y6825x5.aspx>