Министерство образования и науки РФ
Санкт-Петербургский государственный университет аэрокосмического приборостроения
Кафедра Аэрокосмических компьютерных и программных систем

Ф.С. Горбачев, А.Ю. Сыщиков, Ю.Е. Шейнин

Параллельное программирование.

OpenMP: интерфейс для SMP архитектур.

Методические указания к выполнению лабораторных работ.

Санкт-Петебург
2020

Оглавление

Введение

Интерфейс OpenMP предназначен для программирования параллельных вычислений на традиционных алгоритмических языках (C/C++, Fortran), выполняемых на системах с общим пространством памяти (SMP). Его основную часть составляют набор директив, библиотечных функций и переменных окружения. В OpenMP заложена идея “инкрементального распараллеливания”, когда разработчик не создает новую программу, просто добавляет в текст последовательной программы OpenMP-директивы. При этом система программирования OpenMP предоставляет разработчику большие возможности по контролю над поведением параллельного приложения.

Вся программа разбивается на последовательные и параллельные области. Все последовательные области выполняет *главная нить* (*master thread*), порождаемая при запуске программы, а при входе в параллельную область главная *нить* порождает дополнительные нити. После выхода из параллельной области происходит синхронизация и все *нити*, кроме *главной нити*, уничтожаются. После этого продолжается последовательное выполнение кода до очередной параллельной области и т.д. Предполагается, что OpenMP-программа без какой-либо модификации должна работать как на многопроцессорных системах, так и на однопроцессорных.

Следует отметить, что наличие общей памяти не препятствует использованию технологий программирования, разработанных для систем с распределенной памятью.

Первая версия стандарта OpenMP для языков C/C++ появилась в октябре 1998 г., а вторая, и, пока, последняя - в марте 2002. Ее полное описание можно найти на сайте www.openmp.org. В связи с выпуском микропроцессора Pentium 4 HT, поддерживающего технологию Hyper Threading, корпорация Intel разработала реализацию стандарта OpenMP для компиляторов Intel C++. Она позволяет программировать параллельные вычисления, на двух логических процессорах, входящих в состав Pentium 4 HT.

1 Директивы OpenMP

1.1 Общие положения

Директивы OpenMP для C/C++ имеют следующий формат:
#pragma omp <имя директивы> [*список параметров***]**

Пример директивы OpenMP.
#pragma omp parallel default(shared) private(beta,pi)

Как известно, в семействе языков С директива **#pragma** предназначена для управления специфическими возможностями компилятора. Ключевое слово **omp** используется для того, чтобы исключить случайные совпадения имен директив OpenMP с другими именами.

Параметры директив (**directive clauses**) можно условно разделить на две категории. Одни из них влияют на ход вычислительного процесса и позволяют учесть особенности распараллеливаемой задачи и многопроцессорной системы, в которой она будет выполняться. Например, с их помощью можно разрешить создание параллельной области только при выполнении заданного условия, назначить количество нитей в регионе или количество итераций, выполняемых в нитях. Другие параметры, их примерно половина, определяют видимость переменных в параллельном регионе и специальные действия, которые должны выполняться над ними в начале и в конце сферы действия директивы. В таких случаях после имени параметра в круглых скобках указывается список переменных, на которые распространяется действие параметра. Он может содержать только те имена, которые описаны в исходном тексте стандартными средствами C/C++ и используются в ассоциированном с директивой операторе или блоке.

Объектом действия большинства директив является один *оператор* или *блок*, перед которым расположена директива в исходном тексте программы. В описании стандарта OpenMP такие операторы или блоки называются ассоциированными с директивой.

В общем случае, директивы OpenMP можно разделить на 3 типа:

- Директивы для определения параллельной области
- Директивы для распределения вычислений
- Директивы синхронизации.

Состав директив и их краткая характеристика приведены в таблице 1 приложения А.

1.2 Директивы для определения параллельной области.

Фрагмент программы, который делится на параллельно выполняемые нити, называется параллельной областью. В стандарте OpenMP ***нить*** (поток, thread) определена как простая программная компонента, имеющая частные (локальные) переменные, доступ к общим (внешним) переменным и предназначенная для выполнения одним процессором. В зависимости от установленного способа планирования это может быть конкретный или первый свободный процессор.

Для задания параллельной области программы используется директива **parallel**. Это основная директива OpenMP.

Директива **parallel** имеет следующий формат

#pragma omp parallel [*список параметров*...] newline

structured_block

параметры

if (scalar_expression)

private (list)

shared (list)

default (shared | none)

firstprivate (list)

reduction (operator: list)

copyin (list)

Когда поток выполнения достигает директиву **parallel**, создается набор (*team*) из N *нитей*. Входная *нить* этого набора (являющаяся *главной нитью*) имеет номер 0. Код области действия директивы дублируется или разделяется между *нитями* для параллельного выполнения. В конце области обеспечивается синхронизация *нитей* – выполняется ожидание завершения вычислений всех *нитей*; далее все *нити* завершаются, и дальнейшие вычисления продолжает выполнять только главная *нить*.

Количество нитей в параллельной области может задаваться (в порядке убывания старшинства)

1. функцией API **omp_set_num_threads()**
2. через переменную окружения **OMP_NUM_THREADS**
3. реализацией

По умолчанию количество нитей одинаковое. Для активизации динамического режима можно использовать функцию **omp_set_dynamic** или переменную окружения **OMP_DYNAMIC.**

Для условного входа в параллельную область можно использовать параметр директивы **if** – если условие, указанное в нем, не выполняется, нити не создаются.

Пример 1. Использование директивы parallel

```
#include <omp.h>
main () {
 int nthreads, tid;
 /* Создание параллельной области */
#pragma omp parallel private(nthreads, tid)
 {
  /* печать номера потока */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);
  /* Печать количества потоков – только master */
  if (tid == 0) {
   nthreads = omp_get_num_threads();
   printf("Number of threads = %d\n", nthreads);
  }
 }/* Завершение параллельной области  */
```

```
}
```

1.3 Директивы для распределения вычислений внутри параллельной области

1.3.1 Общая характеристика.

Существует 3 директивы для распределения вычислений в параллельной области

- **for** – распределение итераций цикла между *нитями*
- **sections** – распараллеливание раздельных фрагментов кода (функциональное распараллеливание)
- **single** – директива для указания последовательного выполнения кода

В сфере действия директив **for**, **sections** и **single** нельзя использовать все остальные директивы OpenMP, что ограничивает возможности управления вычислительным процессом в параллельном регионе. Для снятия этих ограничений описание региона должно начинаться с директивы **parallel**. Формально сферой ее действия является блок, но он может содержать вложенные блоки и директивы, как формирующие структуру региона, так и управляющие процессом вычислений. Кроме того, директива **parallel** имеет параметры, которые позволяют по усмотрению программиста выбирать количество нитей, выполняемых в параллельном регионе.

При этом, начало выполнения директив по умолчанию не синхронизируется, завершение директив по умолчанию является синхронным.

1.3.2 Директива for

Директива **for** распределяет по имеющимся нитям копии ассоциированного оператора цикла, каждая из которых выполняет некоторую часть от общего количества итераций. Для распараллеливания с помощью **for** необходимо, чтобы параллельная секция уже была активна.

Директива **for** имеет следующий формат:

#pragma omp **for** [*список параметров*] newline
 цикл_for
параметры
schedule (*type* [,*chunk*])
ordered
private (*list*)
firstprivate (*list*)
lastprivate (*list*)
shared (*list*)
reduction (*operator*: *list*)
nowait

Распределение итераций цикла регулируется параметром **schedule**. Он может принимать следующие значения:

- **static** – итерации делятся на блоки по *chunk* итераций и статически разделяются между *нитями*. Если параметр chunk не определен, итерации делятся между *нитями* равномерно и непрерывно.

- **dynamic** – распределение итерационных блоков осуществляется динамически (по умолчанию chunk=1)
- **guided** – размер итерационного блока уменьшает экспоненциально при каждом распределении; chunk определяет минимальный размер блока (по умолчанию chunk=1)
- **runtime** – правило распределения определяется переменной окрущения **OMP_SCHEDULE** (при использовании **runtime** параметр chunk задаваться не должен)

Пример 2. Распараллеливание цикла с помощью директивы for.

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
main () {
 int i, n, chunk;
 float a[NMAX], b[NMAX], c[NMAX];
 /* Инициализация */
 for (i=0; i < NMAX; i++)
  a[i] = b[i] = i * 1.0;
 n = NMAX;
 chunk = CHUNK;
 #pragma omp parallel shared(a,b,c,n,chunk) private(i)
 {
  #pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < n; i++)
    c[i] = a[i] + b[i];
 } /* end of parallel section */
}
```

1.3.3 Директива sections

Директива **sections** формирует нити из отдельных блоков, расположенных в исходной программе последовательно друг за другом. Перед каждым преобразуемым в нить блоком указывается вспомогательная директива **section**.

При этом:

- каждый блок выполняется однократно
- разные блоки выполняются разными потоками
- завершение директивы **sections** по умолчанию синхронизируется
- директивы **section** должны использоваться только в статическом контексте

Директива **sections** имеет следующий формат:
#pragma omp sections *[список параметров...] newline*
{
#pragma omp section *newline*
structured_block
#pragma omp section *newline*
structured_block
}
список параметров
private *(list)*
firstprivate *(list)*
lastprivate *(list)*
reduction *(operator: list)*
nowait

Пример 3 Распараллеливание с помощью sections

```
#include <omp.h>
#define NMAX 1000
main () {
 int i, n;
 float a[NMAX], b[NMAX], c[NMAX];
/* Инициализация */
 for (i=0; i < NMAX; i++)
   a[i] = b[i] = i * 1.0;
 n = NMAX;
 #pragma omp parallel shared(a,b,c,n) private(i)
 {
  #pragma omp sections nowait
  {
    #pragma omp section
      for (i=0; i < n/2; i++)
       c[i] = a[i] + b[i];
    #pragma omp section
      for (i=n/2; i < n; i++)
       c[i] = a[i] + b[i];
  } /* Завершение sections */
 } /* Завершение параллельной области*/
}
```

1.3.4 Директива single

Директива **single** задает блок, который должен быть выполнен только одним потоком; все остальные потоки ожидают завершения выполнения фрагмента (если не указан параметр nowait)

Директива **single** имеет следующий формат:
#pragma omp single [*список параметров*...] newline
structured_block

список параметров
private (list)
firstprivate (list)
nowait

1.3.5 Объединение директив parallel и for(sections)

В случае, если параллельная область содержит только одну директиву for (или sections), возможно объединение этих директив в одну.

Пример 4. Объединение директив parallel и for

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
main ()
{
     int i, n, chunk;
     float a[NMAX], b[NMAX], c[NMAX];
     /* Инициализация */
     for (i=0; i < NMAX; i++)
      a[i] = b[i] = i * 1.0;
     n = NMAX;
     chunk = CHUNK;
     #pragma omp parallel for shared(a,b,c,n) private(i)
schedule(static,chunk)
    for (i=0; i < n; i++)
        c[i] = a[i] + b[i];
}
```

1.4 Классы переменных.

В OpenMP переменные в параллельных областях программы разделяются на два основных класса:

• shared (общие; под именем А все нити видят одну переменную) и

• private (частные; под именем А каждая нить видит свою переменную).

По умолчанию действует простое правило определения видимости переменных в параллельной области. Те из них, которые в исходном тексте программы описаны вне распараллеливаемого оператора или блока, объявляются общими, а описанные внутри оператора или блока - частными. Общие (**shared**) переменные доступны всем нитям, а частные (**private**) создаются для каждой нити только на время ее выполнения. Пример общей переменной - имя массива, элементы которого обрабатываются в теле цикла, а переменная, управляющая повторами цикла, является частной, иначе будет невозможно распараллеливание цикла.

Принятое по умолчанию деление переменных на общие и частные применимо не во всех случаях и тогда приходится явно указывать способ доступа или особенности использования конкретных переменных. Отдельные правила определяют поведение переменных при входе и выходе из параллельной области или параллельного цикла:

firstprivate – частные копии переменной при входе в параллельную область инициализируются значением оригинальной переменной

lastprivate - по окончании параллельно цикла или блока параллельных секций, нить, которая выполнила последнюю итерацию цикла или последнюю секцию блока, обновляет значение оригинальной переменной.

threadprivate - объявляет частными в нитях параллельного региона переменные, описанные во внешнем блоке

copyin - для каждой *нити* параллельной области создаются копии переменных, описанных в директиве **threadprivate**. Копиям присваиваются текущие значения оригиналов. Имена, указанные в списках директивы **threadprivate** и параметра **copyin**, должны совпадать

reduction - ограничивает доступ к оригиналам переменных. В нитях используются копии, а значения оригиналов корректируются после выполнения каждой нити. Ограниченные переменные должны использоваться в качестве операндов указанного перед их списком оператора.

Рассмотрим два случая, которые возможны при распараллеливании оператора цикла.

Предположим, что результаты вычислений, выполняемых в теле цикла, накапливаются в переменной *sum*. По умолчанию она объявляется общей и доступна всем нитям. Конечный результат окажется корректным, но в процессе выполнения задания сразу несколько нитей будут обращаться к переменной *sum* для чтения или записи ее содержимого. Это неизбежно замедлит процесс выполнения задания. Просто объявить переменную *sum* частной нельзя, поскольку будут потеряны результаты вычислений, накопленные в нитях. Специально для такого случая предназначен параметр **reduction**. Если в его списке указать переменную *sum*, то в нитях она будет использоваться как частная, а при выходе из каждой нити накопленный результат будет добавлен к общей переменной **sum**.

Другой пример. Предположим, что в теле цикла для хранения промежуточных результатов вычислений используется переменная *temp*. Если в исходном тексте ее описание расположено вне оператора цикла, то при распараллеливании по умолчанию она будет объявлена общей переменной. В таком случае при выполнении задания вполне вероятны случаи, когда одна из нитей успеет изменить значение *temp*, прежде чем другая использует сформированное в ней значение. Это неизбежно приведет к ошибкам в результатах вычислений. Поэтому все переменные, используемые для хранения промежуточных результатов, следует объявлять частными, либо описывать их непосредственно в теле оператора цикла.

1.5 Директивы синхронизации

С помощью директив синхронизации можно явно определить тот или иной способ синхронизации *нитей*. Некоторые из директив могут использоваться неявно.

OpenMP включает в себя следующие директивы синхронизации:

master - определяет блок кода, который будет выполнен только *главной нитью*

critical - определяет критическую секцию, то есть блок кода, который не должен выполняться одновременно двумя или более нитями

barrier - определяет точку барьерной синхронизации, в которой каждая нить дожидается всех остальных

atomic - определяет переменную в левой части оператора "атомарного" присваивания, которая должна корректно обновляться несколькими нитями

ordered - определяет блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации идут в последовательном цикле. Может использоваться для упорядочения вывода от параллельных нитей

flush - явно определяет точку, в которой реализация должна обеспечить одинаковый вид памяти для всех нитей. Неявно **flush** присутствует в следующих директивах **barrier**, в начале **critical** и **ordered** блоков, в конце **critical**, **parallel**, **sections**, **single** и **ordered** блоков.

2 Переменные окружения

Стандарт OpenMP и его реализация для Intel C++ предусматривают существование четырех переменных окружения (**environment variables**). Они содержат статически определенные величины, которые могут использоваться при формировании структуры параллельной области. Имена переменных набираются заглавными латинскими буквами, а аргументы могут набираться как на верхнем, так и на нижнем регистре. Способ установки значений переменных зависит от конкретной реализации компилятора. Изменение значений переменных при выполнении задания невозможно.

Переменные и их назначение в таблице 3 приложения А.

3 Функции библиотеки runtime

В описании стандарта OpenMP для C/C++ перечислены 17 функций, входящих в состав библиотеки runtime. Наиболее важные из них перечислены в таблице 4 приложения А. Они позволяют в процессе выполнения задания определить и изменить количество нитей, используемых в параллельном регионе, определить номер конкретной нити и количество доступных процессоров в системе.

Для возможности использования библиотечных функций к приложению должен быть подключен заголовочный файл <**omp.h**>. Все четыре функции целочисленные.

4 Компиляция и запуск программ OpenMP

Для компиляции C/C++ программ, использующих OpenMP, достаточно использовать компилятор с поддержкой OpenMP. На данный момент OpenMP поддерживают как все основные компиляторы, такие как Microsoft Compiler, Intel Compiler, GCC, так и некоторые другие, например, CLang.

Задания к лабораторным работам.

4.1 Лабораторная работа №1. Распараллеливание операций над векторами.

4.1.1 Цель работы

Реализовать и распараллелить с помощью технологии OpenMP различные операции над векторами размерности N.

4.1.2 Содержание отчета

Задание.
Исходный текст программы.
Результат работы программы.

4.1.3 Варианты заданий

№ в-та	N	Тип элемента вектора	Тип операции
1	5	Число, с плавающей запятой	Скалярное произведение 2-х векторов
2	7	Знаковый короткий целый	Сложение векторов (a[i]=b[i]+c[i])
3	9	Без знаковый целый	Определение максимального элемента
4	11	Длинное целое	Определение минимального элемента
5	13	Целое	Вычисление суммы элементов
6	5	Целое	Двоичное искл. ИЛИ
7	7	Длинное целое	Определение максимального элемента
8	9	Знаковый короткий целый	Скалярное произведение 2-х векторов
9	11	Без знаковый целый	Сложение векторов (a[i]=b[i]+c[i])
10	13	Целый байтовый	Двоичное И
11	5	Целый байтовый	Двоичное искл. ИЛИ
12	7	Знаковый короткий целый	Двоичное искл. ИЛИ
13	9	Без знаковый целый	Двоичное ИЛИ
14	11	Длинное целое	Скалярное произведение 2-х векторов
15	13	Целое	Определение минимального элемента
16	9	Длинное целое	Определение минимального элемента
17	11	Знаковый короткий целый	Двоичное И
18	13	Целый байтовый	Определение максимального элемента

4.2 Лабораторная работа №2. Распараллеливание вычислений.

4.2.1 Цель работы

Реализовать и распараллелить с помощью OpenMP различные алгоритмы.

4.2.2 Содержание отчета

Задание.
Исходный текст программы.
Результаты работы программы.

4.2.3 Варианты заданий

№ в-та	Размерность задачи	Тип элемента данных	Алгоритм
1	Точность 0.001		Вычисление константы ПИ
2	20x60	Знаковый короткий целый	Сортировка столбцов матрицы по возрастанию
3	70	Без знаковый целый	Сортировка массива методом пузырька
4	80	Длинное целое	Сортировка массива методом QuickSort
5	50	Целое	Сортировка массива методом Шелла.
6	Точность 0.001		Вычисление интеграла функции F(x)=x*x в заданных границах
7	70x90	Длинное целое	Сортировка строк матрицы по возрастанию
8	80x30	Знаковый короткий целый	Сортировка строк матрицы по убыванию
9	90x60	Без знаковый целый	Сортировка столбцов матрицы по убыванию
10	100x50	Число с плавающей запятой	Перемножение матриц
11	90x40	Целый байтовый	Сортировка столбцов матрицы по возрастанию
12	80	Знаковый короткий целый	Сортировка массива методом пузырька
13	70	Без знаковый целый	Сортировка массива методом QuickSort
14	60	Длинное целое	Сортировка массива методом Шелла.
15	Точность 0.001		Вычисление интеграла функции F(x)=x*x*x в заданных границах
16	20x70	Длинное целое	Сортировка строк матрицы по возрастанию
17	100x50	Знаковый короткий целый	Сортировка строк матрицы по убыванию
18	40x60	Целый байтовый	Сортировка столбцов матрицы по убыванию

Библиографический список

Русскоязычные источники:

1. Воеводин Вл.В. Курс лекций "Параллельная обработка данных". http://parallel.ru/vvv/index.html.
2. http://parallel.ru/tech/tech_dev/openmp.html

Англоязычные источники:

1. http://www.openmp.org
2. Introduction to OpenMP. http://www.llnl.gov/computing/tutorials/workshops/workshop/openMP/MAIN.html
3. Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. (2000). *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers.

Приложение А. Справочная информация

Таблица 1. Директивы OpenMP

Директива	Описание
parallel [*параметры*]	Директива имеет декларативный характер и не управляет действиями, выполняемыми в параллельном регионе. Она нужна, например, в тех случаях, когда для распараллеливания региона используется несколько директив формирующих нити или выполняющих другие действия. *Параметры*: **private, firstprivate, shared, default, reduction, copyin, if, num_threads.**
for [*параметры*]	Формирует нити, содержащие копии ассоциированного с директивой цикла типа for. Каждая копия будет выполнять свою часть от общего числа итераций, описанных в исходном операторе цикла. *Параметры*: **private, firstprivate, lastprivate, reduction, ordered, nowait, schedule.**
sections [*параметры*]	Формирует параллельный регион из блоков, расположенных в исходном тексте программы последовательно друг за другом. Перед каждым преобразуемым блоком указывается директива **section**. *Параметры*: **private, firstprivate, lastprivate, reduction, nowait.**
section	Вспомогательная директива, используется только в области действия директивы **sections** для формирования нитей из ассоциированных блоков.
single [*параметры*]	Указывает на то, что в регионе должна выполняться только одна нить, содержащая ассоциированный с директивой блок. Такая нить может, например, изменять значения частных переменных, используемых другими нитями региона. *Параметры*: **private, firstprivate, copyprivate, nowait.**
parallel for [*параметры*]	Сокращенная форма записи для создания параллельного региона, содержащего единственную директиву **for**. Сочетание двух директив увеличивает количество доступных параметров. *Параметры*: **private, firstprivate, lastprivate, shared, default, reduction, ordered, schedule, copyin, if, num_threads.**
parallel sections [*параметры*]	Сокращенная форма записи для создания параллельного региона, содержащего единственную директиву **sections**. Сочетание двух директив увеличивает количество доступных параметров. *Параметры*: **private, firstprivate, lastprivate, shared, default, reduction, copyin, if, num_threads.**
master	Ассоциированный с директивой блок преобразуется в основную нить, с которой начинается выполнение задачи. Выполнение основной нити продолжается до тех пор, пока не встретится первая распараллеливаемая конструкция.
critical [*имя секции*]	Критические секции нужны для разграничения доступа к общему ресурсу, например, к памяти. Все нити параллельного региона ждут завершения выполнения критической секции. Если есть несколько критических секций, то им надо присвоить уникальные имена.
barrier	Указывает точку, в которой организуется ожидание окончания исполнения всех нитей параллельного региона. По умолчанию (если не указан параметр **nowait**) директивы **for, sections** и **single** устанавливают барьер в нужной точке региона.
atomic	Директива действует только на один оператор присваивания. При каждом его выполнении новое значение переменной, указанной в левой части принудительно сохраняется в памяти. Это позволяет

	исключить возможные ошибки при работе с одной переменной в нескольких нитях параллельного региона.
Flush [*список переменных*]	**Только** указанные в списке, или по умолчанию **все** общие переменные подвергаются операции "выравнивание" (flush). При этом из кеша в основную память переписываются переменные, значения которых были изменены. Это же касается и переменных находящихся в регистрах процессоров.
threadprivate (*список переменных*)	Объявляет частными в нитях параллельного региона переменные, описанные во внешнем блоке. Директива указывается во внешнем блоке сразу после описания соответствующих переменных и не влияет на работу с ними вне параллельного региона. См. **copyin**.
ordered	Используется в сфере действия директивы **for** для выделения блока, в котором повторы цикла будут происходить в естественном порядке (как при обычных последовательных вычислениях). У директивы **for** должен быть указан одноименный ключ **ordered**.

Таблица 2. Параметры директив OpenMP

Параметр	Описание
private (список)	Для каждой нити на время ее выполнения создаются копии перечисленных в списке переменных, доступные только в пределах конкретной нити (на конкретном процессоре). Исходные значения копий не определены. Применяется с директивами **parallel**, **for**, **sections**, **single**.
firstprivate (список)	Отличается от параметра **private** тем, что создаваемым копиям присваивается значения, которые имели перечисленные в списке переменные перед входом в параллельный регион. Применяется с директивами **parallel**, **for**, **sections**, **single**.
lastprivate (список)	Отличается от параметра **private** тем, что после выхода из цикла или из последней секции конечные значения перечисленных в списке переменных будут доступны для использования. Применяется с директивами **for**, **sections**.
shared (список)	Описывает те переменные исходного блока, которые будут доступны всем нитям параллельного региона. Другими словами, нити будут работать не с копиями, а с оригиналами переменных. Применяется с директивой **parallel**.
default (shared | none)	Данный параметр действует на все переменные, используемые в параллельном регионе. **default** (**shared**) делает их общими, а **default** (**none**) недоступными. В последнем случае необходимо явное описание каждой переменной, используемой в регионе. Применяется с директивой **parallel**.
reduction (оператор:список)	Ограничивает доступ к оригиналам переменных. В нитях используются копии, а значения оригиналов корректируются после выполнения каждой нити. Ограниченные переменные должны использоваться в качестве операндов указанного перед их списком оператора. Применяется с директивами **parallel**, **for**, **sections**.
nowait	Отменяет барьер, установленный по умолчанию, поэтому после окончания выполнения нити не происходит ожидания окончания выполнения других нитей. Применяется с директивами **for**, **sections**, **single**.
if (условие)	Если условие выполнено (true), то регион распараллеливается, в противном случае (false) все его блоки выполняются одним процессором в естественном порядке. Применяется с директивой **parallel**.

ordered	Указывает на то, что в теле директивы **for** будет использована директива **ordered**, для выделения блока, в котором итерации выполняются в естественном порядке. Применяется с директивой **for**.
schedule (**static** | **dynamic** | **guided** | **runtime**)	При распараллеливании цикла данный параметр позволяет выбрать примерно одинаковое или монотонно уменьшающееся количество итераций в нитях и способ распределения копий цикла между нитями. Применяется с директивой **for**.
copyin (список)	Для каждой нити параллельного региона создаются копии переменных, описанных в директиве **threadprivate**, копиям присваиваются текущие значения оригиналов. Имена, указанные в списках директивы **threadprivate** и параметра **copyin**, должны совпадать. Применяется с директивой **parallel**.

Следующие два параметра отсутствуют в документации к Intel C++ Compiler, но они предусмотрены стандартом OpenMP для C/C++.

copyprivate (список)	После выполнения нити, содержащей конструкцию **single**, новые значения переменных списка **copyprivate** будут доступны всем одноименным частным переменным (**private** и **firstprivate**), описанным в начале параллельного региона и используемым всеми его нитями. Применяется с директивой **single**.
num_threads (целочисленное выражение)	Предназначен для явного задания количества нитей, которое может содержать параллельный регион. По умолчанию выбирается последнее значение, установленное с помощью функции **omp_set_num_threads**, или значение переменной **omp_num_threads**. Применяется с директивой **parallel**.

Таблица 3. Переменные окружения в OpenMP

Переменная	Описание
OMP_SHEDULE	Содержит тип планирования и количество итераций выполняемых в нитях. Вызывается, например, если для директивы **for** указан параметр **shedule (runtime)**. *Значение по умолчанию*: **static**
OMP_NUM_THREADS	Задает количество нитей в параллельном регионе. Может использоваться, например, если в директиве **parallel** не указан параметр **num_threads**. *Значение по умолчанию*: **количество процессоров в системе**.
OMP_DYNAMIC	Разрешает/запрещает динамическое регулирование количества нитей в параллельном регионе. *Значение по умолчанию*: **false**.
OMP_NESTED	Разрешает или запрещает распараллеливание вложенных (внутренних) циклов. *Значение по умолчанию*: **false**.

Таблица 4. Функции библиотеки runtime

Функция	Описание
omp_get_num_threads	Возвращает фактическое количество нитей, существующих в параллельном регионе. При вызове из последовательного региона возвращает значение 1. *Формат*: **int omp_get_num_threads(void);**

omp_set_num_threads	Устанавливает количество нитей, которое может содержать параллельный регион, имеет приоритет над переменной окружения OMP_NUM_THREADS. Для использования функции должно быть разрешено изменение количества нитей в процессе выполнения задания. *Формат*: **int omp_set_num_threads(int NumThreads);**
omp_get_thread_num	Возвращает номер нити из которой вызвана функция. Номер изменяется в пределах от 0 до omp_get_num_threads() - 1. Для основной нити (master thread) он всегда равен нулю. *Формат*: **int omp_get_thread_num(void);**
omp_get_num_procs	Возвращает количество процессоров, доступных для использования нитями параллельного региона, включая логические процессоры, если поддерживается технология Hyper-Threading. *Формат*: **int omp_get_num_procs(void);**