

**Нижегородский государственный
архитектурно-строительный университет
Кафедра информационных систем и технологий**

Введение в OpenMP

по материалам Гергеля В.П., Сысоева А.В.
(Кафедра математического обеспечения ЭВМ, ННГУ)

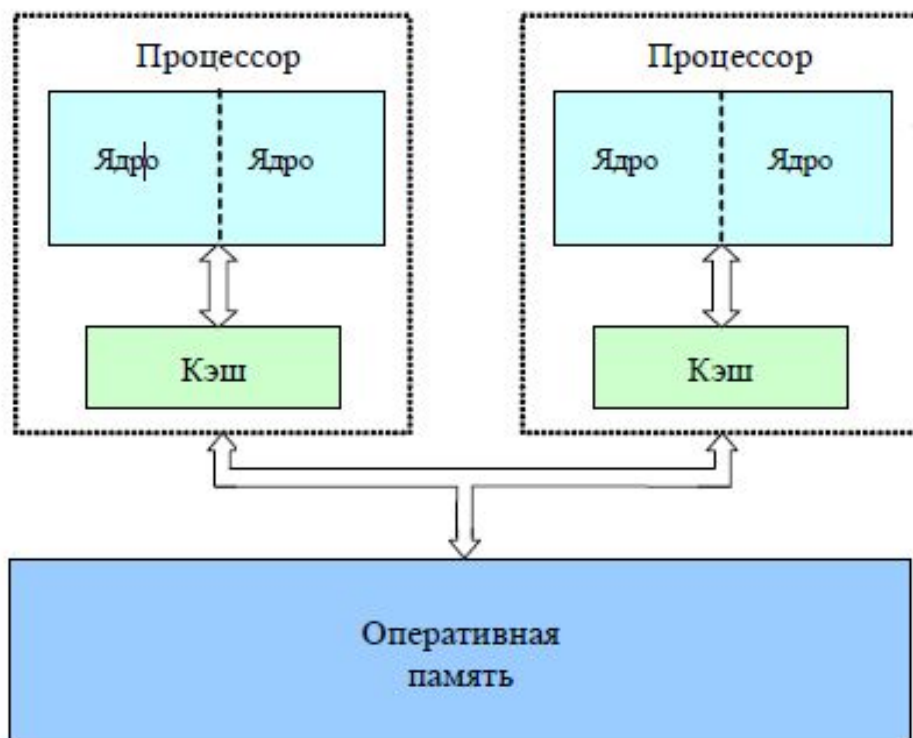
Кислицын Д.И.

Содержание

- ❑ Обзор технологии OpenMP
- ❑ Директивы OpenMP
 - Формат, области видимости, типы
 - Определение параллельной области
 - Управление областью видимости данных
 - Распределение вычислений между потоками
 - Операция редукции
 - Синхронизация
 - Совместимость директив и их параметров
- ❑ Библиотека функций OpenMP
- ❑ Переменные окружения
- ❑ Информационные ресурсы

Обзор технологии OpenMP

- Интерфейс OpenMP задуман как стандарт параллельного программирования для многопроцессорных систем с общей памятью (SMP, NUMA, ...)



В общем вид системы с общей памятью описывается в виде модели параллельного компьютера с произвольным доступом к памяти (*parallel random-access machine – PRAM*)

Обзор технологии OpenMP

Динамика развития стандарта

- ❑ OpenMP Fortran API v1.0 (1997)
- ❑ OpenMP C/C++ API v1.0 (1998)
- ❑ OpenMP Fortran API v2.0 (2000)
- ❑ OpenMP C/C++ API v2.0 (2002)
- ❑ OpenMP C/C++ API v2.5 (2005)

- ❑ Разработкой стандарта занимается организация OpenMP Architecture Review Board, в которую вошли представители крупнейших компаний - разработчиков SMP-архитектур и программного обеспечения.

Обзор технологии OpenMP

- Основания для достижения эффекта – разделяемые для параллельных процессов данные располагаются в общей памяти и для организации взаимодействия не требуется операций передачи сообщений.

Обзор технологии OpenMP

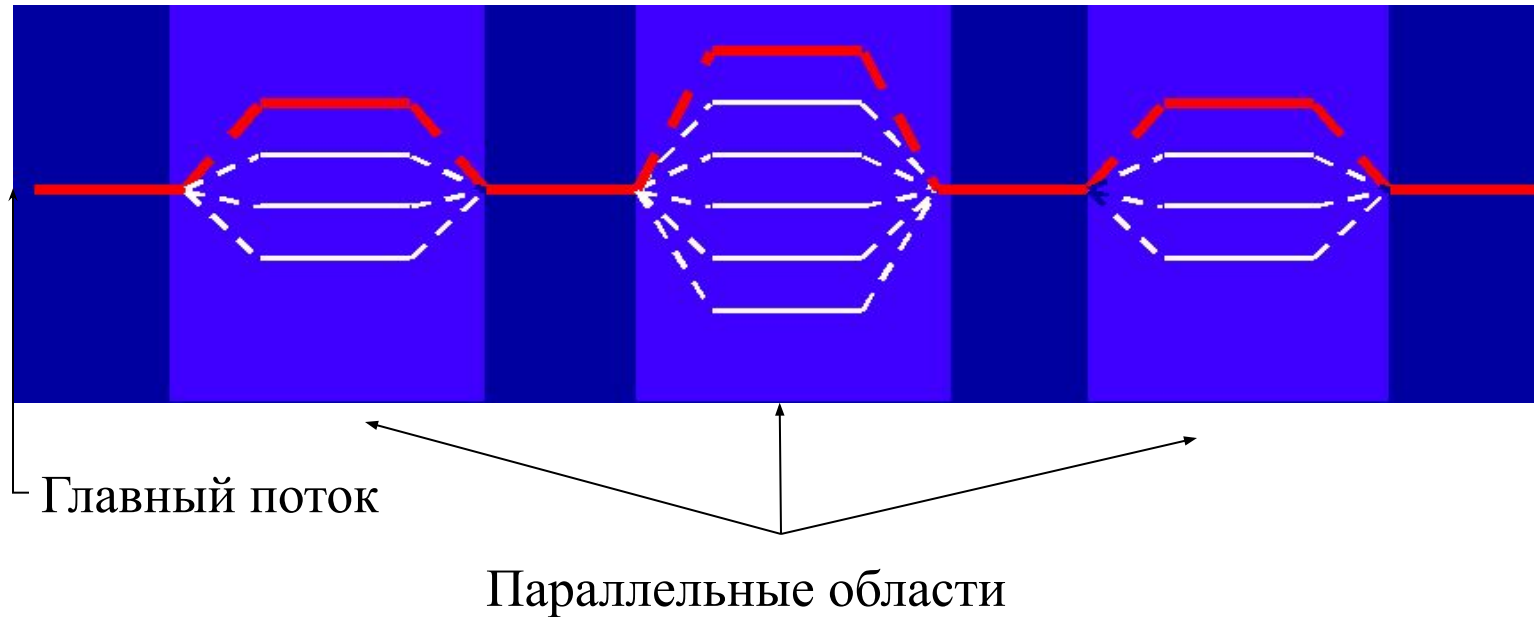
Положительные стороны

- ❑ Поэтапное (инкрементальное) распараллеливание
 - Можно распараллеливать последовательные программы поэтапно, не меняя их структуру
- ❑ Единственность разрабатываемого кода
 - Нет необходимости поддерживать последовательный и параллельный вариант программы, поскольку директивы игнорируются обычными компиляторами (в общем случае)
- ❑ Эффективность
 - Учет и использование возможностей систем с общей памятью
- ❑ Стандартизованность (переносимость), поддержка в наиболее распространенных языках (C/C++, Fortran) и платформах (Windows, Unix)

Обзор технологии OpenMP

Принцип организации параллелизма

- ❑ Использование потоков (общее адресное пространство)
- ❑ Пульсирующий (“вилочный”, fork-join) параллелизм



Обзор технологии OpenMP

Принцип организации параллелизма

- ❑ При выполнении обычного кода (вне параллельных областей) программа выполняется одним потоком (master thread)
- ❑ При появлении директивы `#parallel` происходит создание “команды” (team) потоков для параллельного выполнения вычислений
- ❑ После выхода из области действия директивы `#parallel` происходит синхронизация, все потоки, кроме master, уничтожаются
- ❑ Продолжается последовательное выполнение кода (до очередного появления директивы `#parallel`)

Термины и понятия

- ❑ *Параллельный фрагмент (parallel construct)* – блок программы, управляемый директивой **parallel**; именно параллельные фрагменты, совместно с параллельными областями, представляют параллельно-выполняемую часть программы.
- ❑ *Параллельная область (parallel region)* – параллельно выполняемые участки программного кода, динамически-возникающие в результате вызова функций из параллельных фрагментов.
- ❑ *Параллельная секция (parallel section)* – часть параллельного фрагмента, выделяемая для параллельного выполнения при помощи директивы **section**.

Обзор технологии OpenMP

Структура

- ❑ Набор директив компилятора
- ❑ Библиотека функций
- ❑ Набор переменных окружения

- ❑ Изложение материала будет проводиться на примере C/C++

Директивы OpenMP

Формат записи директив

□ Формат

```
#pragma omp имя_директивы [clause,...]
```

□ Пример

```
#pragma omp parallel default(shared) \  
    private(beta,pi)
```

Пример показывает также, что для задания директивы может быть использовано несколько строк программы – признаком наличия продолжения является знак обратного слеша "\".

Действие директивы распространяется, как правило, на следующий в программе оператор, который может быть, в том числе, и структурированным блоком.

Директивы OpenMP

Типы директив

- ❑ Определение параллельной области
- ❑ Разделение работы
- ❑ Синхронизация

Директивы OpenMP

Определение параллельной области

- Директива `parallel` (основная директива OpenMP)

Когда основной поток выполнения достигает директиву `parallel`, создается набор (`team`) потоков; входной поток является основным потоком этого набора (`master thread`) и имеет номер 0

Код области дублируется или разделяется между потоками для параллельного выполнения

В конце области обеспечивается синхронизация потоков – выполняется ожидание завершения вычислений всех потоков; далее все потоки завершаются – дальнейшие вычисления продолжает выполнять только основной поток

Директивы OpenMP

Определение параллельной области

- Пример использования директивы `parallel`

```
#include <stdio.h>
#include <omp.h>
int 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
}
```

Директивы OpenMP

Определение параллельной области

- ❑ Формат директивы parallel

```
#pragma omp parallel [clause ...] newline  
structured_block
```

- ❑ Возможные параметры (clause)

```
if (scalar_expression)  
num_threads(integer-expression)  
private (list)  
firstprivate (list)  
default (shared | none)  
shared (list)  
copyin (list)  
reduction (operator: list)
```

Директивы OpenMP

Определение параллельной области

- ❑ Количество потоков (по убыванию старшинства)
 - `num_threads(N)`
 - `omp_set_num_threads()`
 - `OMP_NUM_THREADS`
 - Число, равное количеству процессоров, которое “видит” операционная система
- ❑ Параметр (clause) `if` – если условие в `if` не выполняется, то процессы не создаются

Директивы OpenMP

Управление областью видимости данных

- Управление областью видимости обеспечивается при помощи параметров (clause) директив

private, firstprivate, lastprivate, shared, reduction, ...

которые определяют, какие соотношения существуют между переменными последовательных и параллельных фрагментов выполняемой программы

Директивы OpenMP

Управление областью видимости данных

- ❑ Параметр **shared** определяет список переменных, которые будут общими для всех потоков параллельной области; правильность использования таких переменных должна обеспечиваться программистом `shared (list)`
- ❑ Параметр **default** может отменить действие правила по умолчанию (`default(none)`) или восстановить правило, что по умолчанию: *переменные программы являются общими* (`default(shared)`).
- ❑ Параметр **private** определяет список переменных, которые будут локальными для каждого потока; переменные создаются в момент формирования потоков параллельной области; начальное значение переменных является неопределенным `private (list)`

Директивы OpenMP

Управление областью видимости данных

Начальные значения локальных переменных не определены, а конечные значения теряются при завершении потоков

- ❑ Параметр **firstprivate** позволяет создать локальные переменные потоков, которые перед использованием инициализируются значениями исходных переменных `firstprivate (list)`
- ❑ Параметр **lastprivate** позволяет создать локальные переменные потоков, значения которых запоминаются в исходных переменных после завершения параллельной области (используются значения потока, выполнившего последнюю итерацию цикла или последнюю секцию) `lastprivate (list)`

Определение времени выполнения параллельной программы

- Получение текущего момента времени выполнения фрагментов кода параллельных программ обеспечивается при помощи функции

double omp_get_wtime(void)

Функция возвращает количество секунд, прошедших от некоторого определенного момента времени в прошлом.

```
double t1, t2, dt;  
t1 = omp_get_wtime ();  
...  
t2 = omp_get_wtime ();  
dt = t2 - t1;
```

Директивы OpenMP

Распределение вычислений между потоками

- ❑ Существует 3 директивы для распределения вычислений в параллельной области
 - DO / for – распараллеливание циклов
 - sections – распараллеливание отдельных фрагментов кода (функциональное распараллеливание)
 - single – директива для указания последовательного выполнения кода

- ❑ Начало выполнения директив по умолчанию не синхронизируется

- ❑ Завершение директив по умолчанию является синхронным

Директивы OpenMP

Распределение вычислений между потоками

❑ Формат директивы for

```
#pragma omp for [clause ...] newline  
for loop
```

❑ Возможные параметры (clause)

```
private(list)  
firstprivate(list)  
lastprivate(list)  
reduction(operator: list)  
ordered  
schedule(kind[, chunk_size])  
nowait
```

Директивы OpenMP

Распределение вычислений между потоками

- Распределение итераций в директиве `for` регулируется параметром (clause) `schedule`
 - `static` – итерации делятся на блоки по `chunk` итераций и статически разделяются между потоками; если параметр `chunk` не определен, итерации делятся между потоками равномерно и непрерывно
 - `dynamic` – распределение итерационных блоков осуществляется динамически (по умолчанию `chunk=1`)
 - `guided` – размер итерационного блока уменьшается экспоненциально при каждом распределении; `chunk` определяет минимальный размер блока (по умолчанию `chunk=1`)
 - `runtime` – правило распределения определяется переменной `OMP_SCHEDULE` (при использовании `runtime` параметр `chunk` задаваться не должен) Так, например, для задания динамического способа при размере блока итераций 3, следует определить:

```
setenv OMP_SCHEDULE "dynamic,3"
```

Директивы OpenMP

Распределение вычислений между потоками

□ Пример использования директивы for

```
#include <stdio.h>
#include <omp.h>
#define NMAX 100
int main () {
    int i, j;
    double a[NMAX][NMAX], sum;
    for (i=0; i < NMAX; i++)
        for (j=0; j < NMAX; j++)
            a[i][j] = (i+j);
    #pragma omp parallel for shared(a) private(i,j,sum)
    for (i=0; i < NMAX; i++) {
        sum = 0;
        for (j=0; j < NMAX; j++)
            sum += a[i][j];
        printf ("Summa of row elements %d is %f\n", i, sum);
    } /* Завершение параллельного фрагмента */
}
```


Директивы OpenMP

Распределение вычислений между потоками

- Пример динамического распределения итераций между потоками

```
#include <stdio.h>
#include <omp.h>
#define NMAX 100
#define CHUNK 10
int main () {
    int i, j;
    double a[NMAX][NMAX], sum;
    for (i=0; i < NMAX; i++)
        for (j=0; j < NMAX; j++)
            a[i][j] = (i+j);
    #pragma omp parallel for shared(a) private(i,j,sum) \
        schedule (dynamic, CHUNK)
    for (i=0; i < NMAX; i++) {
        sum = 0;
        for (j=i; j < NMAX; j++)
            sum += a[i][j];
        printf ("Summa of row elements %d is %f\n", i, sum);
    } /* Завершение параллельного фрагмента */
}
```

Директивы OpenMP

Распределение вычислений между потоками

□ Управление порядком выполнения вычислений

Если же для ряда действий в цикле необходимо сохранить первичный порядок вычислений, который соответствует последовательному выполнению итераций в последовательной программе, то желаемого результата можно добиться при помощи директивы **ordered** (при этом для директивы **for** должен быть указан параметр **ordered**). Параметр **ordered** управляет порядком выполнения только тех действий, которые выделены директивой **ordered**.

```
#pragma omp parallel for shared(a) private(i,j,sum) \  
                schedule (dynamic, CHUNK) ordered  
{  
  for (i=0; i < NMAX; i++) {  
    sum = 0;  
    for (j=i; j < NMAX; j++)  
      sum += a[i][j];  
    #pragma omp ordered  
    printf ("Сумма элементов строки %d равна %f\n", i, sum);  
  } /* Завершение параллельного фрагмента */
```

Директивы OpenMP

Распределение вычислений между потоками

- Синхронизация вычислений по окончании выполнения цикла

По умолчанию, все потоки, прежде чем перейти к выполнению дальнейших вычислений, ожидают окончания выполнения итераций цикла даже если некоторые из них уже завершили свои вычисления – конец цикла представляет собой некоторый барьер, который потоки могут преодолеть только все вместе.

Можно отменить указанную синхронизацию, указав параметр ***nowait*** в директиве ***for*** – тогда потоки могут продолжить вычисления за пределами цикла, если для них нет итераций цикла для выполнения.

Директивы OpenMP

Операция редукции

- Параметр `reduction` определяет список переменных, для которых выполняется операция редукции
 - перед выполнением параллельной области для каждого потока создаются копии этих переменных,
 - потоки формируют значения в своих локальных переменных
 - при завершении параллельной области над всеми локальными значениями выполняются необходимые операции редукции, результаты которых запоминаются в исходных (глобальных) переменных

`reduction (operator: list)`

Директивы OpenMP

Операция редукции

□ Правила записи параметра reduction

Возможный формат записи выражения

$x = x \text{ op expr}$

$x = \text{expr op } x$

$x \text{ binop} = \text{expr}$

$x++$, $++x$, $x--$, $--x$

- x должна быть скалярной переменной
- expr не должно ссылаться на x
- op (operator) должна быть неперегруженной операцией вида
 $+$, $-$, $*$, $/$, $\&$, \wedge , $|$, $\&\&$, $\|\|$
- binop должна быть неперегруженной операцией вида
 $+$, $-$, $*$, $/$, $\&$, \wedge , $|$

Директивы OpenMP

Операция редукции

□ Пример использования параметра reduction

```
total = 0;
#pragma omp parallel for shared(a) private(i,j,sum) \
    reduction (+:total)
for (i=0; i < NMAX; i++) {
sum = 0;
for (j=i; j < NMAX; j++)
sum += a[i][j];
printf ("Сумма элементов строки %d равна %f\n",i,sum);
total = total + sum;
} /* Завершение параллельного фрагмента */
printf ("Общая сумма элементов матрицы равна %f\n",total);
```

Использование общей переменной `total` без создания локальных копий является неправильным, т.к. без обеспечения взаимоисключения возникает ситуация гонки потоков и итоговый результат может быть ошибочным.

Директивы OpenMP

Распределение вычислений между потоками

□ Параметр `if` директивы `parallel`

```
#include <omp.h>
#define NMAX 1000
#define LIMIT 100
main () {
int i, j, sum;
float a[NMAX][NMAX];
<инициализация данных>
#pragma omp parallel for shared(a) private(i,j,sum) \
                        if (NMAX>LIMIT)
{
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);
} /* Завершение параллельного фрагмента */
}
```

Директивы OpenMP

Распределение вычислений между потоками

□ Формат директивы sections

```
#pragma omp sections [clause ...] newline
{
    #pragma omp section newline
    structured_block
    #pragma omp section newline
    structured_block
}
```

□ Возможные параметры (clause)

```
private(list)
firstprivate(list)
lastprivate(list)
reduction(operator: list)
nowait
```


Директивы OpenMP

Распределение вычислений между потоками

- Директива `sections` – распределение вычислений для отдельных фрагментов кода
 - фрагменты выделяются при помощи директивы `section`
 - каждый фрагмент выполняется однократно
 - разные фрагменты выполняются разными потоками
 - завершение директивы по умолчанию синхронизируется
 - директивы `section` должны использоваться только в статическом контексте

Директивы OpenMP

Распределение вычислений между потоками

□ Пример использования директивы sections

```
total = 0;
#pragma omp parallel shared(a,b) private(i,j)
{
  #pragma omp sections
  {
    #pragma omp section
    /* Вычисление сумм элементов строк и общей суммы */
    for (i=0; i < NMAX; i++) {
      sum = 0;
      for (j=0; j < NMAX; j++)
        sum += a[i][j];
      printf ("Summa of row elements %d is %f\n",i,sum);
      total = total + sum;
    }
    #pragma omp section
    /* Копирование матрицы */
    for (i=0; i < NMAX; i++)
      for (j=0; j < NMAX; j++)
        b[i][j] = a[i][j];
  }
  /* Завершение параллельного фрагмента */
  printf ("Total summa is %f\n",total);
}
```

Директивы OpenMP

Распределение вычислений между потоками

□ Объединение директив parallel и for/sections

```
total = 0;
#pragma omp parallel sections shared(a,b) private(i,j)
{
#pragma omp section
/* Вычисление сумм элементов строк и общей суммы */
for (i=0; i < NMAX; i++) {
    sum = 0;
    for (j=0; j < NMAX; j++)
        sum += a[i][j];
    printf ("Summa of row elements %d is %f\n",i,sum);
    total = total + sum;
}
#pragma omp section
/* Копирование матрицы */
for (i=0; i < NMAX; i++)
    for (j=0; j < NMAX; j++)
        b[i][j] = a[i][j];
} /* Завершение параллельного фрагмента */
printf ("Total summa is %f\n",total);
}
```

Директивы OpenMP

Распределение вычислений между потоками

□ Формат директивы `single`

```
#pragma omp single [clause ...] newline  
    structured_block
```

□ Возможные параметры (clause)

```
private(list)  
firstprivate(list)  
copyprivate(list)  
nowait
```

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***).

Директивы OpenMP

Синхронизация

- 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)

```
#pragma omp master newline  
structured_block
```

- Директива `critical` определяет фрагмент кода, который должен выполняться только одним потоком в каждый текущий момент времени (критическая секция)

```
#pragma omp critical [name] newline  
structured_block
```

Директивы OpenMP

Синхронизация

- Пример использования директивы **critical**

```
#include <omp.h>
main() {
    int x;
    x = 0;
    #pragma omp parallel shared(x)
    {
        #pragma omp critical
        x = x + 1;
    } // end of parallel section
}
```

Директивы OpenMP

Синхронизация

- Директива `barrier` – определяет точку синхронизации, которую должны достигнуть все процессы для продолжения вычислений (директива должна быть вложена в блок)

```
#pragma omp barrier newline
```

Директивы OpenMP

Синхронизация

- Директива `atomic` – определяет переменную, доступ к которой (чтение/запись) должна быть выполнена как неделимая операция

```
#pragma omp atomic newline  
expression
```

- Возможный формат записи выражения `expression`
`x binop = expr`, `x++`, `++x`, `x--`, `--x`
`x` должна быть скалярной переменной
`expr` не должно ссылаться на `x`
`binop` должна быть неперегруженной операцией вида
`+`, `-`, `*`, `/`, `&`, `^`, `|`, `>>`, `<<`

Директивы OpenMP

Синхронизация

□ Пример использования директивы **atomic**

```
total = 0;
#pragma omp parallel for shared(a) private(i,j,sum)
  for (i=0; i < NMAX; i++) {
    sum = 0;
    for (j=0; j < NMAX; j++)
      sum += a[i][j];
    printf ("Summa of row elements %d is %f\n",i,sum);
    #pragma omp atomic
    total += sum;
  } /* Завершение параллельного фрагмента */
printf ("Total summa is %f\n",total);
```

Директивы OpenMP

Синхронизация

- Директива `flush` – определяет точку синхронизации, в которой системой должно быть обеспечено единое для всех процессов состояние памяти (т.е. если потоком какое-либо значение извлекалось из памяти для модификации, измененное значение обязательно должно быть записано в общую память)

```
#pragma omp flush (list) newline
```

- Если указан список `list`, то восстанавливаются только указанные переменные
- Директива `flush` неявным образом присутствует в директивах `barrier`, `critical`, `ordered`, `parallel`, `for`, `sections`, `single`

Директивы OpenMP

Совместимость директив и их параметров

Clause	Directive					
	PARALLEL	DO/for	SECTIONS	SINGLE	PARALLEL DO/for	PARALLEL SECTIONS
IF	•				•	•
PRIVATE	•	•	•	•	•	•
SHARED	•	•			•	•
DEFAULT	•				•	•
FIRSTPRIVATE	•	•	•	•	•	•
LASTPRIVATE		•	•		•	•
REDUCTION	•	•	•		•	•
COPYIN	•				•	•
SCHEDULE		•			•	
ORDERED		•			•	
NOWAIT		•	•	•		

Сводный перечень директив OpenMP

Директива	Описание
parallel [<параметр>...]	Директива определения параллельного фрагмента в коде программы (подраздел 4.2). Параметры: <i>if, private, shared, default, firstprivate, reduction, copyin, num_threads</i>
for [<параметр>...]	Директива распараллеливания циклов (подраздел 4.2). Параметры: <i>private, firstprivate, lastprivate, reduction, ordered, nowait, schedule</i>
sections [<параметр>...]	Директива для выделения программного кода, который далее будет разделен на параллельно выполняемые параллельные секции (подраздел 4.6). Выделение параллельных секций осуществляется при помощи директивы section . Параметры: <i>private, firstprivate, lastprivate, reduction, nowait</i>
section	Директива выделения параллельных секций – должна располагаться в блоке директивы sections (подраздел 4.6).
single [<параметр>...]	Директива для выделения программного кода в параллельном фрагменте, исполняемого только одним потоком (п. 4.7.1). Параметры: <i>private, firstprivate, copyprivate, nowait</i>

Сводный перечень директив OpenMP

Директива	Описание
<code>parallel for</code> [<параметр>...]	Объединенная форма директив <code>parallel</code> и <code>for</code> (подраздел 4.2). Параметры: <code>private</code> , <code>firstprivate</code> , <code>lastprivate</code> , <code>shared</code> , <code>default</code> , <code>reduction</code> , <code>ordered</code> , <code>schedule</code> , <code>copyin</code> , <code>if</code> , <code>num_threads</code>
<code>parallel sections</code> [<параметр>...]	Объединенная форма директив <code>parallel</code> и <code>sections</code> (подраздел 4.6). Параметры: <code>private</code> , <code>firstprivate</code> , <code>lastprivate</code> , <code>shared</code> , <code>default</code> , <code>reduction</code> , <code>copyin</code> , <code>if</code> , <code>num_threads</code>
<code>master</code>	Директива для выделения программного кода в параллельном фрагменте, исполняемого только основным (<i>master</i>) потоком (п. 4.7.1).
<code>critical [(name)]</code>	Директива для определения критических секций (п. 4.5.2).
<code>barrier</code>	Директива для барьерной синхронизации потоков (п. 4.7.2).
<code>atomic</code>	Директива для определения атомарной (неделимой) операции (п. 4.5.1).
<code>flush [list]</code>	Директива для синхронизации состояния памяти (п. 4.7.3).
<code>threadprivate (list)</code>	Директива для определения постоянных локальных переменных потоков (п. 4.7.4).
<code>ordered</code>	Директивы управления порядком вычислений в распараллеливаемом цикле (п. 4.3.2). При использовании данной директивы в директиве <code>for</code> должен быть указан одноименный параметр <code>ordered</code>

Сводный перечень параметров директив OpenMP

Параметр	Описание
<code>private (list)</code>	Параметр для создания локальных копий для перечисленных в списке переменных для каждого имеющегося потока (п. 4.4.1). Исходные значения копий не определены. Директивы: <code>parallel</code> , <code>for</code> , <code>sections</code> , <code>single</code>
<code>firstprivate (list)</code>	Тоже что и параметр <code>private</code> и дополнительно инициализация создаваемых копий значениями, которые имели перечисленные в списке переменные перед началом параллельного фрагмента (п. 4.4.1). Директивы: <code>parallel</code> , <code>for</code> , <code>sections</code> , <code>single</code>
<code>lastprivate (list)</code>	Тоже что и параметр <code>private</code> и дополнительно запоминание значений локальных переменных после завершения параллельного фрагмента (п. 4.4.1). Директивы: <code>for</code> , <code>sections</code>
<code>shared (list)</code>	Параметр для определения общих переменных для всех имеющихся потоков (п. 4.4.1). Директивы: <code>parallel</code>
<code>default (shared none)</code>	Параметр для установки правила по умолчанию на использование переменных в потоках (п. 4.4.1). Директивы: <code>parallel</code>

Сводный перечень параметров директив OpenMP

Параметр	Описание
<code>reduction</code> (<i>operator: list</i>)	Параметр для задания операции редукции (п. 4.4.2). Директивы: <code>parallel</code> , <code>for</code> , <code>sections</code>
<code>nowait</code>	Параметр для отмены синхронизации при завершении директивы. Директивы: <code>for</code> , <code>sections</code> , <code>single</code>
<code>if</code> (<i>expression</i>)	Параметр для задания условия, только при выполнении которого осуществляется создание параллельного фрагмента (п. 4.3.4). Директивы: <code>parallel</code>
<code>ordered</code>	Параметр для задания порядка вычислений в распараллеливаемом цикле (п. 4.3.2). Директивы: <code>for</code>
<code>schedule</code> (<i>type</i> [, <i>chunk</i>])	Параметр для управления распределением итераций распараллеливаемого цикла между потоками (п. 4.3.1). Директивы: <code>for</code>
<code>copyin</code> (<i>list</i>)	Параметр для инициализации постоянных переменных потоков (п. 4.7.4). Директивы: <code>parallel</code>
<code>copyprivate</code> (<i>list</i>)	Копирование локальных переменных потоков после выполнения блока директивы <code>single</code> (п. 4.7.1). Директивы: <code>single</code>
<code>num_treads</code>	Параметр для задания количества создаваемых потоков в параллельной области (п. 4.7.5). Директивы: <code>parallel</code>

Библиотека функций OpenMP

- Позволяет назначить максимальное число потоков для использования в следующей параллельной области (если это число разрешено менять динамически). **Вызывается из последовательной области программы**

```
void omp_set_num_threads(int num_threads)
```

- Возвращает максимальное число потоков

```
int omp_get_max_threads(void)
```

- Возвращает фактическое число потоков в параллельной области программы

```
int omp_get_num_threads(void)
```


Библиотека функций OpenMP

- ❑ Возвращает номер потока

```
int omp_get_thread_num(void)
```

- ❑ Возвращает число процессоров, доступных приложению

```
int omp_get_num_procs(void)
```

- ❑ Возвращает true, если вызвана из параллельной области программы

```
int omp_in_parallel(void)
```

- ❑ Возвращает количество сек., прошедших от некоторого определенного момента времени в прошлом

```
double omp_get_wtime(void)
```

- ❑ возвращает время в сек. между двумя последовательными показателями времени аппаратного таймера

```
double omp_get_wtick(void)
```

Библиотека функций OpenMP

- ❑ Разрешение (`dynamic=true`) динамического режима и его отключение (`dynamic=false`)

```
void omp_set_dynamic (int dynamic);
```

- ❑ Возвращает *true*, если динамический режим разрешён

```
int omp_get_dynamic(void);
```

- ❑ Разрешение (`nested=true`) вложенного параллелизма и его отключение (`nested=false`)

```
int omp_set_nested(int nested);
```

- ❑ Возвращает *true*, если вложенный параллелизм разрешён

```
int omp_get_nested(void);
```

Переменные окружения

- ❑ `OMP_SCHEDULE` – определяет способ распределения итераций в цикле, если в директиве `for` использована клауза `schedule(runtime)`
- ❑ `OMP_NUM_THREADS` – определяет число нитей для исполнения параллельных областей приложения
- ❑ `OMP_DYNAMIC` – разрешает или запрещает динамическое изменение числа нитей
- ❑ `OMP_NESTED` – разрешает или запрещает вложенный параллелизм

Информационные ресурсы

- ❑ Гергель В.П. Раздел "Параллельное программирование с использованием OpenMP». Учебный курс "Введение в методы параллельного программирования». - Н. Новгород, 2007
- ❑ www.openmp.org
- ❑ Что такое OpenMP – http://parallel.ru/tech/tech_dev/openmp.html
- ❑ Introduction to OpenMP - www.llnl.gov/computing/tutorials/workshops/workshop/openMP/MAIN.html
- ❑ Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. Parallel Programming in OpenMP. – Morgan Kaufmann Publishers, 2000