OpenMP

Параллельные архитектуры

- Системы с распределенной памятью.
- Системы с общей (разделяемой) памятью
- Комбинированные системы

OpenMP

- Для систем с общей памятью (SMP-системах (SSMP, ccNUMA и других) в модели общей памяти (*shared memory model*)
  Набор директив компилятора и библиотечных функций
- C/C++, Fortran
- Поддерживается производителями аппаратуры (Intel, HP, SGI, Sun, IBM), разработчиками
  компиляторов (Intel, Microsoft, KAI, PGI, PSR, APR, Absoft)
- Версия 5.1. (Ноябрь 202) https://www.openmp.org/specifications/

Концепция

- За основу берётся последовательная программа, а для создания её параллельной версии пользователю предоставляется набор директив, функций и переменных окружения. Предполагается, что создаваемая параллельная программа будет переносимой между различными компьютерами с разделяемой памятью, поддерживающими OpenMP API.
- Многопоточность: master “главный” и slave “подчиненный”
- SMP-системах (SSMP, ccNUMA и других) в модели общей памяти (*shared memory model*)

Концепция

- OpenMP можно рассматривать как высокоуровневую надстройку над Pthreads из POSIX или аналогичными библиотеками нитей
- в OpenMP используется терминология и модель программирования, близкая к Pthreads
- Любой UNIX-процесс состоит из нескольких нитей управления, которые имеют общее адресное пространство, но разные потоки команд и раздельные стеки.
- В простейшем случае процесс состоит из одной нити.
- Нити иногда называют также потоками, легковесными процессами, LWP (light-weight processes)

Инкрементальное программирование

- программист постепенно находит участки в программе, содержащие ресурс параллелизма, с помощью предоставляемых механизмов делает их параллельными, а
затем переходит к анализу следующих участков

Компиляция программы

- в Visual C++ **- /openmp**
- заголовочный файл **omp.h**

- макрос **_OPENMP**

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

Пример 1а. Условная компиляция на языке Си.

Модель параллельной программы

- *SPMD*-модель *(Single Program Multiple Data)*

1. Последовательная область – 1 процесс
2. Параллельная область – порождаются еще процессы
3. Последовательная область – 1 процесс
4. …

- Порождение нитей быстрое

Важно

- Равномерная загрузка
- Доступ к общим данным
- Директивы синхронизации

Директивы и функции

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

3 категории:

- определение параллельной области,
- распределение работы,
- синхронизация

Опции (*clause*)

Выполнение программы

- **double omp_get_wtime(void);**
  double omp_get_wtick(void);

```
#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 2а. Работа с системными таймерами на языке Си.

Параллельные и последовательные области

- Параллельная область задаётся при помощи директивы **parallel** (**parallel ... end parallel**).

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   printf("Последовательная область 1\n");
#pragma omp parallel
   {
      printf("Параллельная область\n");
   }
   printf("Последовательная область 2\n");
}
```

Пример 3а. Параллельная область на языке Си.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   int count = 0;
#pragma omp parallel reduction (+: count)
   {
      count++;
      printf("Текущее значение count: %d\n", count);
   }
   printf("Число нитей: %d\n", count);
}
```

Пример 4а. Опция reduction на языке Си.

Переменные среды

- **void omp_set_num_threads(int num);**

- **OMP_NUM_THREADS** нитей

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   omp_set_num_threads(2);
#pragma omp parallel num_threads(3)
   {
      printf("Параллельная область 1\n");
   }
#pragma omp parallel
   {
      printf("Параллельная область 2\n");
   }
}
```

Пример 5а. Функция omp_set_num_threads() и опция num_threads на языке Си.

Переменные среды

- **OMP_DYNAMIC** – изменение количества нитей
- **void omp_set_dynamic(int num);**

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   printf("Значение OMP_DYNAMIC: %d\n", omp_get_dynamic());
   omp_set_dynamic(1);
   printf("Значение OMP_DYNAMIC: %d\n", omp_get_dynamic());
#pragma omp parallel num_threads(128)
   {
#pragma omp master
      {
         printf("Параллельная область, %d нитей\n",
                omp_get_num_threads());
      }
   }
}
```

Пример 6а. Функции omp_set_dynamic() и omp_get_dynamic() на языке Си.

Переменные среды

- **int omp_get_max_threads(void);** максимально допустимое число нитей для использования в следующей параллельной области
- **OMP_NESTED** – вложенность переменных среды
- **void omp_set_nested(int nested)**

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int n;
   omp_set_nested(1);
#pragma omp parallel private(n)
   {
      n=omp_get_thread_num();
#pragma omp parallel
      {
         printf("Часть 1, нить %d - %d\n", n,
                omp_get_thread_num());
      }
   }
   omp_set_nested(0);
#pragma omp parallel private(n)
   {
      n=omp_get_thread_num();
#pragma omp parallel
      {
         printf("Часть 2, нить %d - %d\n", n,
                omp_get_thread_num());
      }
   }
}
```

Пример 7а. Вложенные параллельные области на языке Си.

Переменные среды

- **int omp_get_nested(void);** значение переменной **OMP_NESTED**
- **int omp_in_parallel(void);** возвращает **1** если она была вызвана из активной параллельной области программы

```
#include <stdio.h>
#include <omp.h>
void mode(void){
   if(omp_in_parallel()) printf("Параллельная область\n");
   else printf("Последовательная область\n");
}
int main(int argc, char *argv[])
{
   mode();
#pragma omp parallel
   {
#pragma omp master
      {
         mode();
      }
   }
}
```

Пример 8а. Функция omp_in_parallel() на языке Си.

Директива *single*

- Если в параллельной области какой-либо участок кода должен быть выполнен лишь один раз
  #pragma omp single [опция **[[,]** опция**]...]**

```
#include <stdio.h>
int main(int argc, char *argv[])
{
#pragma omp parallel
   {
      printf("Сообщение 1\n");
#pragma omp single nowait
      {
         printf("Одна нить\n");
      }
      printf("Сообщение 2\n");
   }
}
```

Пример 9а. Директива single и опция nowait на языке Си.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int n;
#pragma omp parallel private(n)
   {
      n=omp_get_thread_num();
      printf("Значение n (начало): %d\n", n);
#pragma omp single copyprivate(n)
      {
         n=100;
      }
      printf("Значение n (конец): %d\n", n);
   }
}
```

Пример 10а. Опция copyprivate на языке Си.

Директивы **master** (**master ... end master**)

- выделяют участок кода, который будет выполнен только нитью-мастером

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   int n;
#pragma omp parallel private(n)
   {
      n=1;
#pragma omp master
      {
         n=2;
      }
      printf("Первое значение n: %d\n", n);
#pragma omp barrier
#pragma omp master
      {
         n=3;
      }
      printf("Второе значение n: %d\n", n);
   }
}
```

Пример 11а. Директива master на языке Си.

Модель данных. Private

- общей для всех нитей области памяти, так и локальной области памяти для каждой нити
- **shared** (общие; все нити видят одну и ту же переменную)
- **private** (локальные, приватные; каждая нить видит свой экземпляр данной переменной).
- «гонка данных» (*data race*)

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

int main(int argc, char *argv[])
{
   int n=1;
   printf("n в последовательной области (начало): %d\n", n);
#pragma omp parallel private(n)
   {
      printf("Значение n на нити (на входе): %d\n", n);
      /* Присвоим переменной n номер текущей нити */
      n=omp_get_thread_num();
      printf("Значение n на нити (на выходе): %d\n", n);
   }
   printf("n в последовательной области (конец): %d\n", n);
}
```

Пример 12а. Опция private на языке Си.

Shared

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int i, m[10];
   printf("Массив m в начале:\n");
   /* Заполним массив m нулями и напечатаем его */
   for (i=0; i<10; i++){
      m[i]=0;
      printf("%d\n", m[i]);
   }
#pragma omp parallel shared(m)
   {
      /* Присвоим 1 элементу массива m, номер которого
      совпадает с номером текущий нити */
      m[omp_get_thread_num()]=1;
   }
   /* Ещё раз напечатаем массив */
   printf("Массив m в конце:\n");
   for (i=0; i<10; i++) printf("%d\n", m[i]);
}
```

Пример 13а. Опция shared на языке Си.

Firstprivate

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int n=1;
   printf("Значение n в начале: %d\n", n);
#pragma omp parallel firstprivate(n)
   {
      printf("Значение n на нити (на входе): %d\n", n);
      /* Присвоим переменной n номер текущей нити */
      n=omp_get_thread_num();
      printf("Значение n на нити (на выходе): %d\n", n);
   }
   printf("Значение n в конце: %d\n", n);
}
```

Пример 14а. Опция firstprivate на языке Си.

threadprivate

```
#include <stdio.h>
#include <omp.h>
int n;
#pragma omp threadprivate(n)
int main(int argc, char *argv[])
{
   int num;
   n=1;
#pragma omp parallel private (num)
   {
      num=omp_get_thread_num();
      printf("Значение n на нити %d (на входе): %d\n", num, n);
      /* Присвоим переменной n номер текущей нити */
      n=omp_get_thread_num();
      printf("Значение n на нити %d (на выходе): %d\n", num, n);
   }
   printf("Значение n (середина): %d\n", n);
#pragma omp parallel private (num)
   {
      num=omp_get_thread_num();
      printf("Значение n на нити %d (ещё раз): %d\n", num, n);
   }
}
```

Пример 15а. Директива threadprivate на языке Си.

copyin

```
#include <stdio.h>
int n;
#pragma omp threadprivate(n)
int main(int argc, char *argv[])
{
   n=1;
#pragma omp parallel copyin(n)
   {
      printf("Значение n: %d\n", n);
   }
}
```

Пример 16а. Опция copyin на языке Си.

Распределение работы. Низкоуровневое распараллеливание

- нити от **0** до **N-1**
- **omp_get_thread_num()** позволяет нити получить свой уникальный номер в текущей параллельной области
- **omp_get_num_threads()** позволяет нити получить количество нитей в текущей параллельной области

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int count, num;
#pragma omp parallel
   {
      count=omp_get_num_threads();
      num=omp_get_thread_num();
      if (num == 0) printf("Всего нитей: %d\n", count);
      else printf("Нить номер %d\n", num);
   }
}
```

Пример 17а. Функции omp_get_num_threads() и omp_get_thread_num() на языке Си.

Параллельные циклы

- **#pragma omp for [**опция **[[,]** опция**]...]**

- **for([**целочисленный тип**] i =** инвариант цикла**;**
  i {<,>,=,<=,>=} инвариант цикла**;**
  i {+,-}= инвариант цикла**)**

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int A[10], B[10], C[10], i, n;
/* Заполним исходные массивы */
   for (i=0; i<10; i++){ A[i]=i; B[i]=2*i; C[i]=0; }
#pragma omp parallel shared(A, B, C) private(i, n)
   {
      /* Получим номер текущей нити */
      n=omp_get_thread_num();
#pragma omp for
      for (i=0; i<10; i++)
      {
         C[i]=A[i]+B[i];
         printf("Нить %d сложила элементы с номером %d\n",
                n, i);
      }
   }
}
```

Пример 18а. Директива for на языке Си.

Параллельные циклы. Опция schedule

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int i;
#pragma omp parallel private(i)
   {
#pragma omp for schedule (static)
//#pragma omp for schedule (static, 1)
//#pragma omp for schedule (static, 2)
//#pragma omp for schedule (dynamic)
//#pragma omp for schedule (dynamic, 2)
//#pragma omp for schedule (guided)
//#pragma omp for schedule (guided, 2)
      for (i=0; i<10; i++)
      {
         printf("Нить %d выполнила итерацию
                omp_get_thread_num(), i);
         sleep(1);
      }
   }
}
```

Пример 19а. Опция schedule на языке Си.

i	static	static, 1	static, 2	dynamic	dynamic, 2	guided	guided, 2
0	0	0	0	0	0	0	0
1	0	1	0	1	0	2	0
2	0	2	1	2	1	1	1
3	1	3	1	3	1	3	1
4	1	0	2	1	2	1	2
5	1	1	2	3	2	2	2
6	2	2	3	2	3	3	3
7	2	3	3	0	3	0	3
8	3	0	0	1	3	0	0
9	3	1	0	0	3	3	0

Таблица 1. Распределение итераций по нитям.

Параллельные циклы. Опция schedule

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int i;
#pragma omp parallel private(i)
   {
#pragma omp for schedule (static, 6)
//#pragma omp for schedule (dynamic, 6)
//#pragma omp for schedule (guided, 6)
      for (i=0; i<200; i++)
      {
         printf("Нить %d выполнила итерацию %d\n",
                omp_get_thread_num(), i);
         sleep(1);
      }
   }
}
```

Пример 20a. Опция schedule на языке Си.

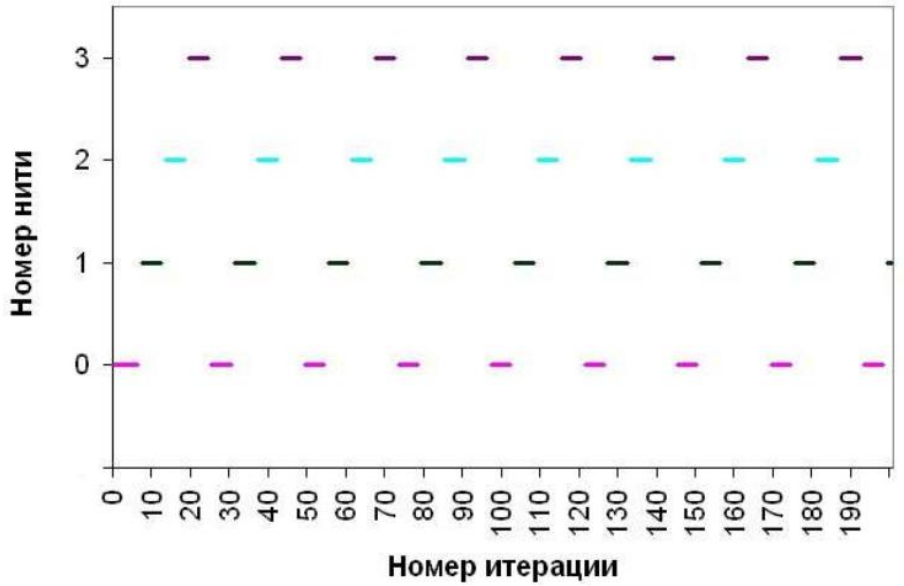

Рис.1a. Распределение итераций по нитям для (static, 6)

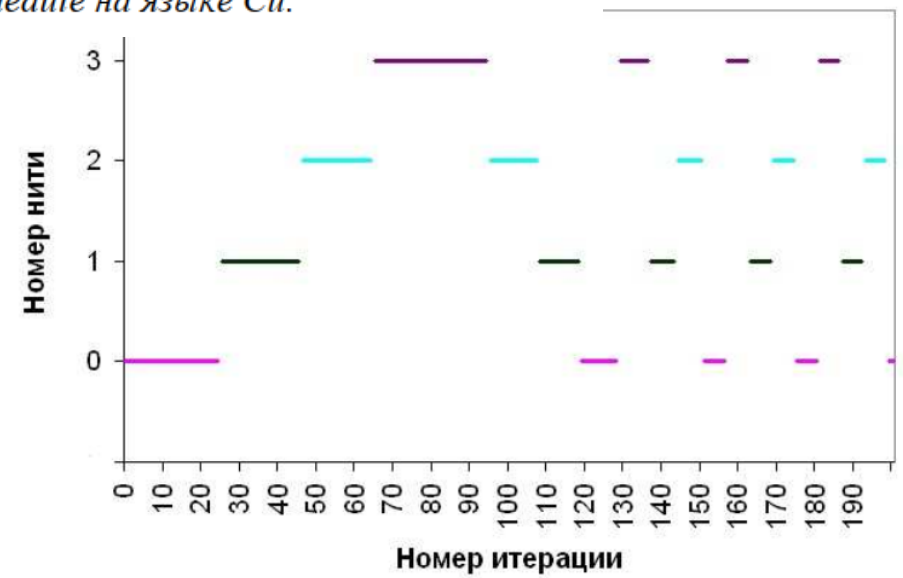

Рис.1c. Распределение итераций по нитям для (guided, 6)

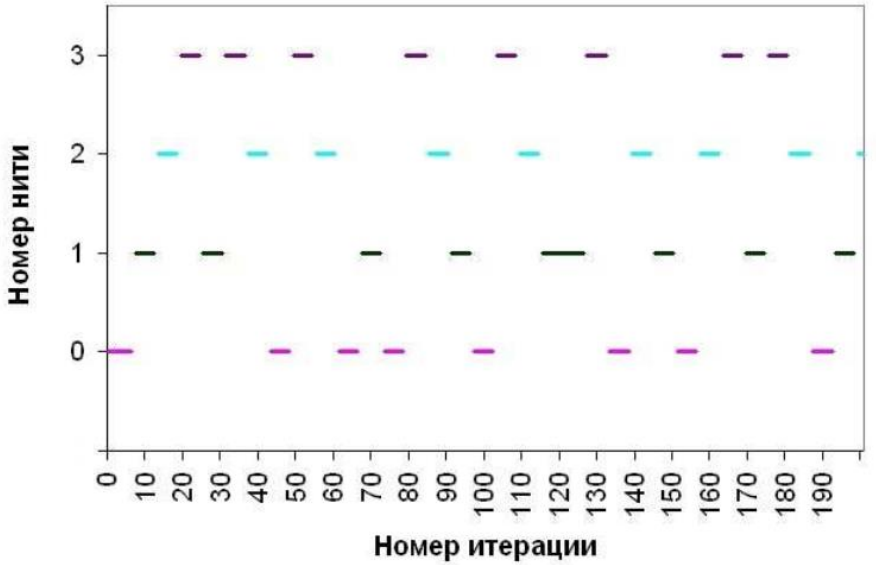

Рис.1b. Распределение итераций по нитям для (dynamic, 6)

OMP_SCHEDULE

- Изменить значение переменной **OMP_SCHEDULE** из программы
  void omp_set_schedule(omp_sched_t type, int chunk);

- **typedef enum omp_sched_t {**
  omp_sched_static = 1,
  omp_sched_dynamic = 2,
  omp_sched_guided = 3,
  omp_sched_auto = 4
  } omp_sched_t;
- **omp_get_schedule()** пользователь может узнать текущее значение переменной **OMP_SCHEDULE**

Параллельные секции

- Директива **sections** (**sections ... end sections**) используется для задания конечного (неитеративного) параллелизма.
  #pragma omp sections [опция **[[,]** опция**]...]**

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int n;
#pragma omp parallel private(n)
   {
      n=omp_get_thread_num();
#pragma omp sections
      {
#pragma omp section
         {
            printf("Первая секция, процесс %d\n", n);
         }
#pragma omp section
         {
            printf("Вторая секция, процесс %d\n", n);
         }
#pragma omp section
         {
            printf("Третья секция, процесс %d\n", n);
         }
      }
      printf("Параллельная область, процесс %d\n", n);
   }
}
```

Пример 21а. Директива sections на языке Си.

Параллельные секции

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int n=0;
#pragma omp parallel
   {
#pragma omp sections lastprivate(n)
      {
#pragma omp section
         {
            n=1;
         }
#pragma omp section
         {
            n=2;
         }
#pragma omp section
         {
            n=3;
         }
      }
      printf("Значение n на нити %d: %d\n",
             omp_get_thread_num(), n);
   }
   printf("Значение n в последовательной области: %d\n", n);
}
```

Пример 22а. Опция lastprivate на языке Си.

Задачи *(tasks)*

- Директива **task** (**task ... end task**) применяется для выделения отдельной независимой задачи.
  #pragma omp task [опция **[[,]** опция**]...]**
- Для гарантированного завершения в точке вызова всех запущенных задач используется директива **taskwait**.
  #pragma omp taskwait

Синхронизация. Барьер

- **#pragma omp barrier**
  Нити, выполняющие текущую параллельную область, дойдя до этой директивы, останавливаются и ждут, пока все нити не дойдут до этой точки программы, после чего разблокируются и продолжают работать дальше. Кроме того, для разблокировки необходимо, чтобы все синхронизируемые нити завершили все порождённые ими задачи (**task**).

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
#pragma omp parallel
   {
      printf("Сообщение 1\n");
      printf("Сообщение 2\n");
#pragma omp barrier
      printf("Сообщение 3\n");
   }
}
```

Пример 24а. Директива barrier на языке Си.

Синхронизация. Директива ***ordered***

- Директивы **ordered** (**ordered ... end ordered**) определяют блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации идут в последовательном цикле.
  #pragma omp ordered

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int i, n;
#pragma omp parallel private (i, n)
   {
      n=omp_get_thread_num();
#pragma omp for ordered
      for (i=0; i<5; i++)
      {
         printf("Нить %d, итерация %d\n", n, i);
#pragma omp ordered
         {
            printf("ordered: Нить %d, итерация %d\n", n, i);
         }
      }
   }
}
```

Пример 25а. Директива ordered и опция ordered на языке Си.

Синхронизация. Критические секции

- С помощью директив **critical** (**critical ... end critical**) оформляется критическая секция программы.
  #pragma omp critical [(<имя_критической_секции**>)]**

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int n;
#pragma omp parallel
   {
#pragma omp critical
      {
         n=omp_get_thread_num();
         printf("Нить %d\n", n);
      }
   }
}
```

Пример 26а. Директива critical на языке Си.

Синхронизация. Директива *atomic*

- Частым случаем использования критических секций на практике является обновление общих переменных. Например, если переменная **sum** является общей и оператор вида **sum=sum+expr** находится в параллельной области программы, то при одновременном выполнении данного оператора несколькими нитями можно получить некорректный результат. Чтобы избежать такой ситуации можно воспользоваться механизмом критических секций или специально предусмотренной для таких случаев директивой **atomic**. **#pragma omp atomic**

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int count = 0;
#pragma omp parallel
      {
#pragma omp atomic
         count++;
      }
      printf("Число нитей: %d\n", count);
}
```

Пример 27а. Директива atomic на языке Си.

Синхронизация. ЗамкИ

- В качестве замков используются общие целочисленные переменные (размер должен быть достаточным для хранения адреса).

Состояния замка

- неинициализированный,
- разблокированный
- заблокированный.

Синхронизация. ЗамкИ

- простые замки
  - **void omp_init_lock(omp_lock_t *lock);**
  - **void omp_destroy_lock(omp_lock_t *lock);**
  - **void omp_set_lock(omp_lock_t *lock);**
  - **void omp_unset_lock(omp_lock_t *lock);**
  - **int omp_test_lock(omp_lock_t *lock);**
- множественные замки
  - **void omp_init_nest_lock(omp_nest_lock_t *lock);**
  - **void omp_destroy_nest_lock(omp_nest_lock_t *lock);**
  - **void omp_set_nest_lock(omp_nest_lock_t *lock);**
  - **void omp_unset_nest_lock(omp_lock_t *lock);**
  - **int omp_test_nest_lock(omp_lock_t *lock);**
  - Коэффициент захваченности *(nesting count)*

```
#include <stdio.h>
#include <omp.h>
omp_lock_t lock;
int main(int argc, char *argv[])
{
   int n;
   omp_init_lock(&lock);
#pragma omp parallel private (n)
   {
      n=omp_get_thread_num();
      omp_set_lock(&lock);
      printf("Начало закрытой секции, нить %d\n", n);
      sleep(5);
      printf("Конец закрытой секции, нить %d\n", n);
      omp_unset_lock(&lock);
   }
   omp_destroy_lock(&lock);
}
```

Пример 28а. Использование замков на языке Си.

Синхронизация. ЗамкИ

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   omp_lock_t lock;
   int n;
   omp_init_lock(&lock);
#pragma omp parallel private (n)
   {
      n=omp_get_thread_num();
      while (!omp_test_lock (&lock))
      {
         printf("Секция закрыта, нить %d\n", n);
         sleep(2);
      }
      printf("Начало закрытой секции, нить %d\n", n);
      sleep(5);
      printf("Конец закрытой секции, нить %d\n", n);
      omp_unset_lock(&lock);
   }
   omp_destroy_lock(&lock);
}
```

Пример 29а. Функция omp_test_lock() на языке Си.

Синхронизация. Директива *flush*

- Поскольку в современных параллельных вычислительных системах может
использоваться сложная структура и иерархия памяти, пользователь должен
иметь гарантии того, что в необходимые ему моменты времени все нити будут видеть единый согласованный образ памяти. Именно для этих целей и
предназначена директива **flush**.
#pragma omp flush [(список**)]**

Примеры программ

```
#include <stdio.h>
double f(double y) {return(4.0/(1.0+y*y));}
int main()
{
   double w, x, sum, pi;
   int i;
   int n = 1000000;
   w = 1.0/n;
   sum = 0.0;
#pragma omp parallel for private(x) shared(w)\
            reduction(+:sum)
   for(i=0; i < n; i++)
   {
      x = w*(i-0.5);
      sum = sum + f(x);
   }
   pi = w*sum;
   printf("pi = %f\n", pi);
}
```

Пример 30а. Вычисление числа Пи на языке Си.

Примеры программ

```
#include <stdio.h>
#include <omp.h>
#define N 4096
double a[N][N], b[N][N], c[N][N];
int main()
{
   int i, j, k;
   double t1, t2;
// инициализация матриц
   for (i=0; i<N; i++)
      for (j=0; j<N; j++)
         a[i][j]=b[i][j]=i*j;
   t1=omp_get_wtime();
// основной вычислительный блок
#pragma omp parallel for shared(a, b, c) private(i, j, k)
   for(i=0; i<N; i++){
      for(j=0; j<N; j++){
         c[i][j] = 0.0;
         for(k=0; k<N; k++) c[i][j]+=a[i][k]*b[k][j];
      }
   }
   t2=omp_get_wtime();
   printf("Time=%lf\n", t2-t1);
}
```

Пример 31а. Перемножение матриц на языке Си.