

Стандарт OpenMP

Лекция 3

Информационные ресурсы

- www.openmp.org
- http://parallel.ru/tech/tech_dev/openmp.html
- www.llnl.gov/computing/tutorials/workshops/workshop/openMP/MAIN.html
- Chandra, R., Menon R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. (2000). *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers.

Стратегия подхода

- OpenMP – стандарт параллельного программирования для многопроцессорных систем с общей памятью.
- Модели параллельного компьютера с произвольным доступом к памяти:
- PRAM – parallel random-access machine

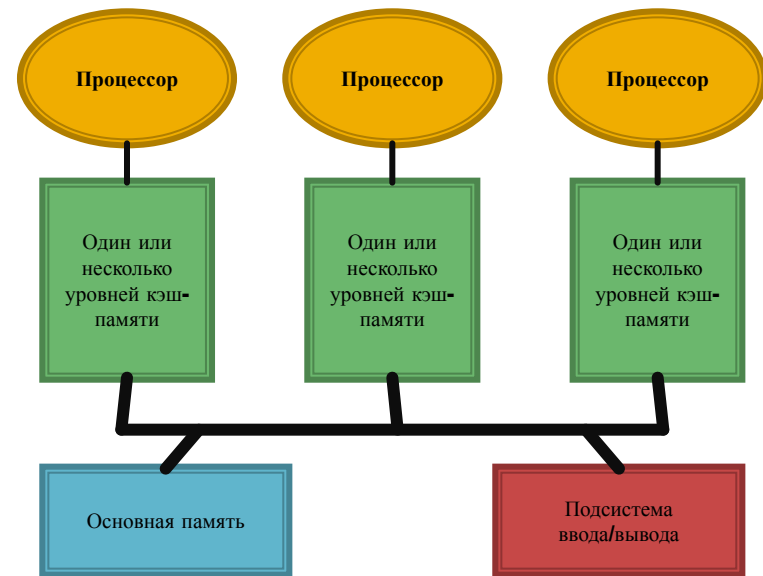


Рис. 1. Модель компьютера

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++/ Fortran API v2.5 (2005)
- OpenMP C/C++/ Fortran API v3.0 (2008)
- OpenMP C/C++/ Fortran API v4.0 (2013)

- Разработкой занимается OpenMP ARB

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++/ Fortran API v2.5 (2005)
- OpenMP C/C++/ Fortran API v3.0 (2008)
- OpenMP C/C++/ Fortran API v4.0 (2013)

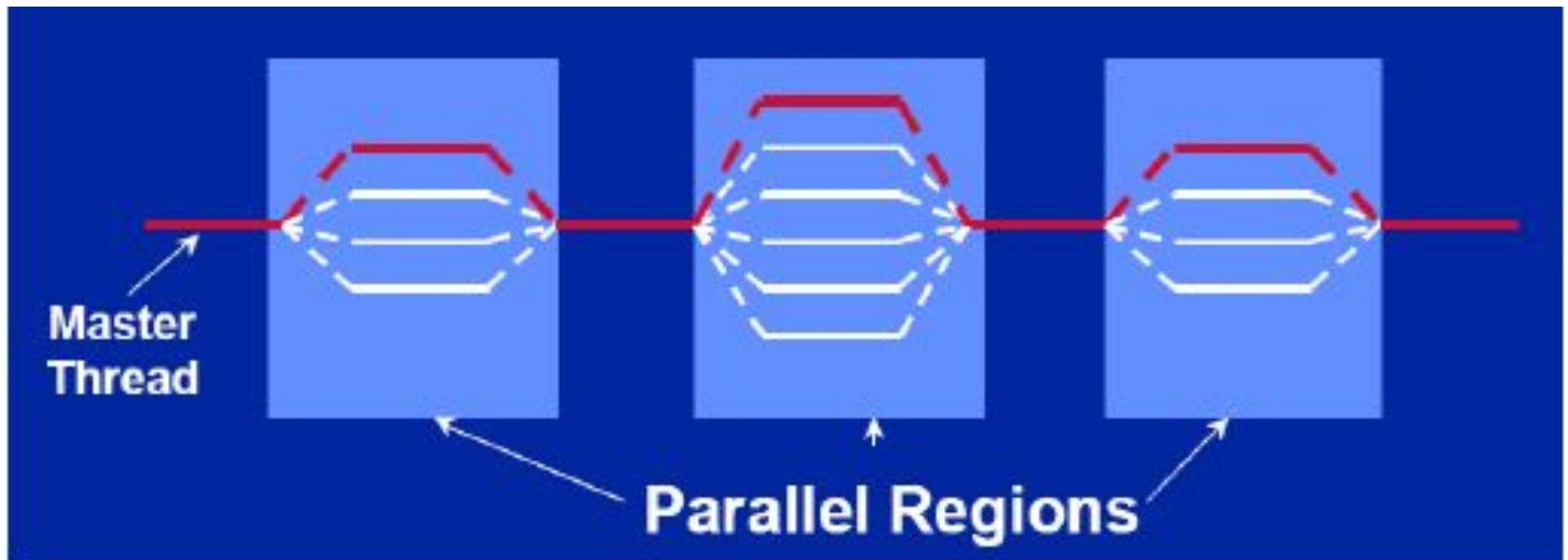
- Разработкой занимается OpenMP ARB

Достоинства

- Поэтапное (инкрементальное) распараллеливание
- Единственность разрабатываемого кода
- Эффективность
- Стандартизированность

Принцип организации параллелизма

- Использование потоков
- Пульсирующий («вилочный») параллелизм



Структура OpenMP:

- Набор директив
- Библиотека функций
- Набор переменных окружения

Директивы OpenMP

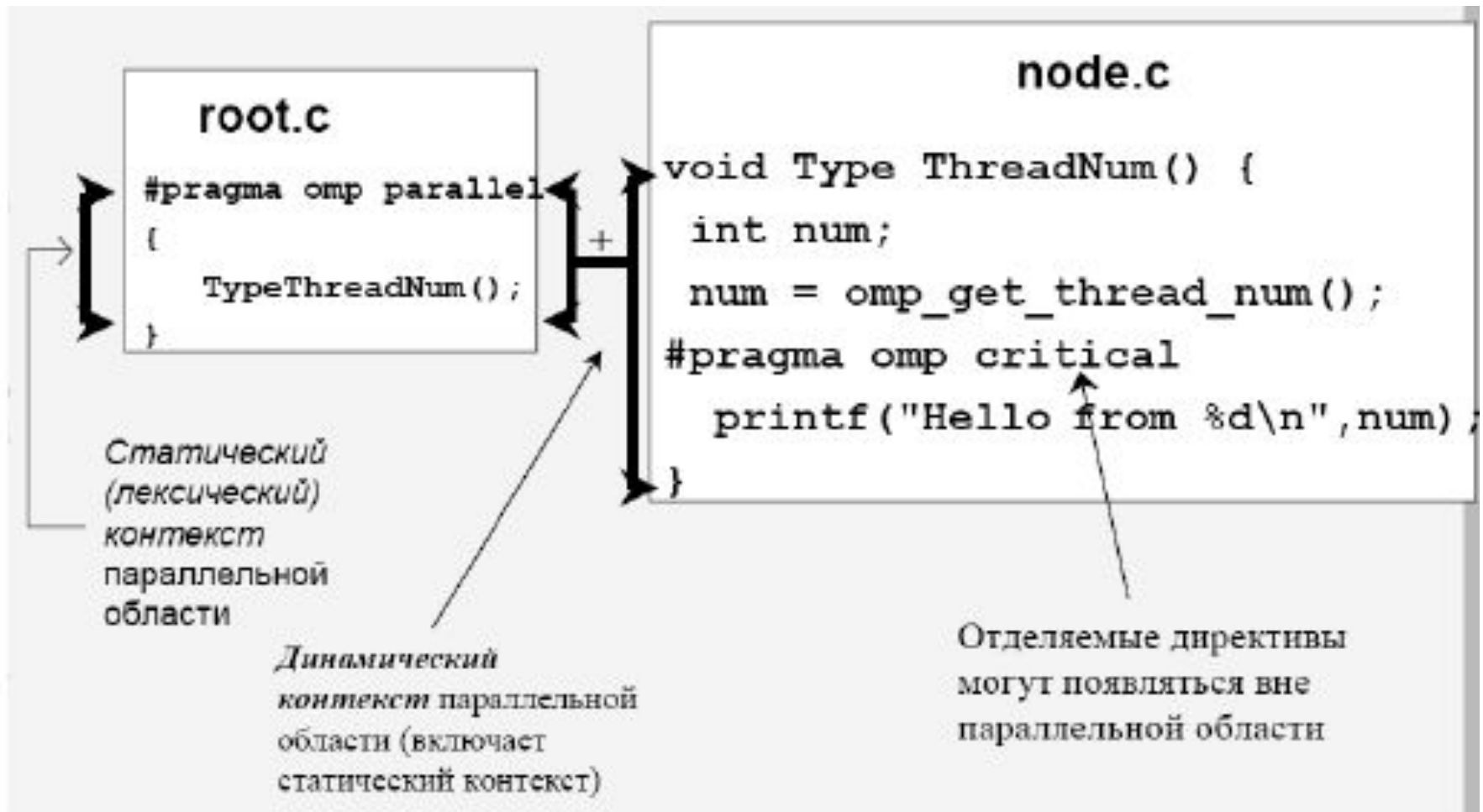
- Формат

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

- Пример

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

Области видимости директив



Типы директив

- Определение параллельной области;
- Разделение работы;
- Синхронизация.

Определение параллельной области

- Директива **parallel**:

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

clause

if (scalar_expression)

private (list)

shared (list)

default (shared | none)

firstprivate (list)

reduction (operator:list)

copyin (list)

Определение параллельной области

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[]) {
    int nthreads, tid;
    #pragma omp parallel private (nthreads, tid) {
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
    }
}
```

Распределение вычислений

- DO/for – распараллеливание циклов
- sections – распараллеливание отдельных фрагментов кода
- single – директива последовательного выполнения кода

Синхронным является только завершение выполнения директив

Директива DO/for

- Директива **DO/for**:
#pragma omp for [clause ...]
for_loop
clause
schedule (type [,chunk])
ordered
private (list)
firstprivate (list)
lastprivate (list)
shared (list)
reduction (operator: list)
nowait

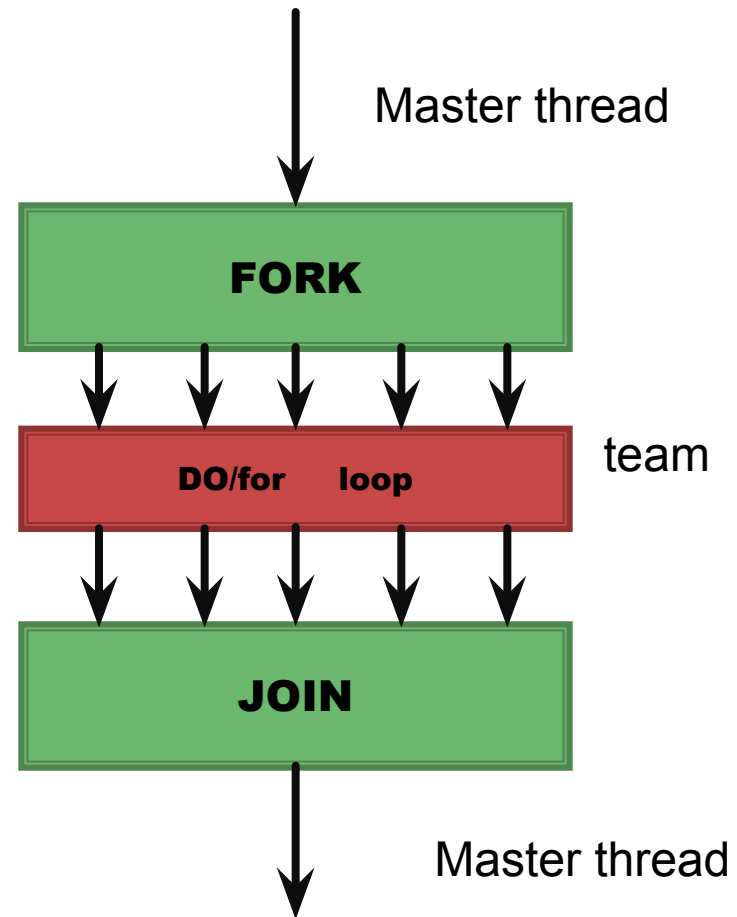


Рис. 4. Модель выполнения

Директива DO/for

```
#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);
        } } }
```


Директива sections

- Директива **section**:
`#pragma omp sections [clause ...]`
{
 `#pragma omp section`
 structured_block...
}
- clause**
private (list)
firstprivate (list)
lastprivate (list)
reduction (operator: list)
nowait

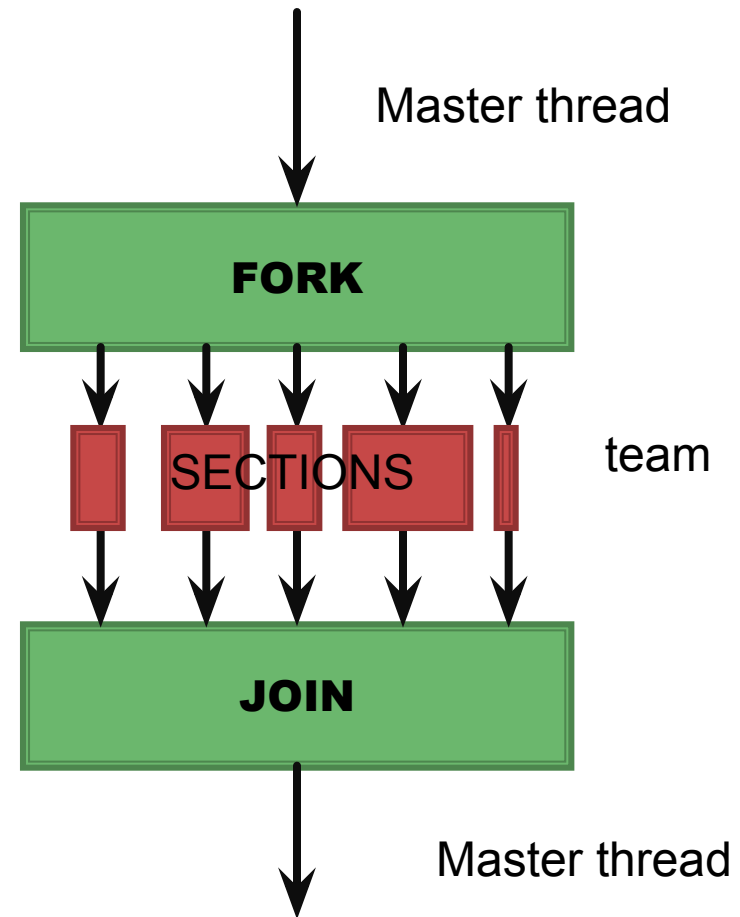


Рис. 5. Модель выполнения

Директива sections

```
#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);
}
```

Директива `single`

- Директива **single**:
`#pragma omp single [clause ...]`
{
 `#pragma omp section`
 `structured_block...`
}
clause
 `private (list)`
 `firstprivate (list)`
 `nowait`

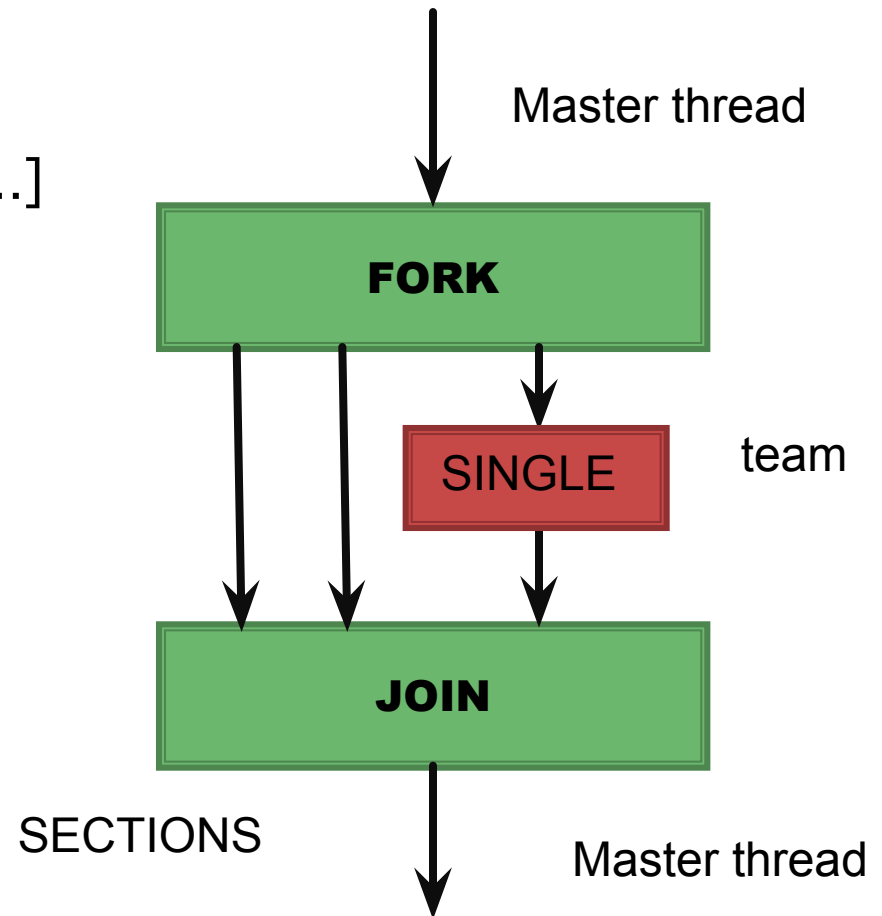


Рис. 5. Модель выполнения

Директива `master`

```
#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);
    } }
```

Директива `critical`

```
#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("Нумь %d\n", n);
        }
    }
}
```

Директива `barrier`

```
#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");
    }
}
```

Директива `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);
}
```

Директива flush

```
#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);
}
```


Директива `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);
            }
        }
    } }
```

Управление областью видимости

if (scalar_expression)
shared (list)
private (list)

clause:

firstprivate (list)
lastprivate (list)
reduction (operator: list)
default (shared | none)

Параметр reduction

- Возможный формат записи:

$x = x \text{ op } expr$

$x = expr \text{ op } x$

$x \text{ binop} = expr$

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

- *x* – скалярная переменная

- *$expr$* не ссылается на *x*

- *op* не перегружен: $+$, $-$, $*$, $\&$, \wedge , $|$, $\&\&$, $||$

- *$binop$* не перегружен: $+$, $-$, $*$, $\&$, \wedge , $|$

Совместимость директив и параметров

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

Библиотека функций OpenMP

- `void omp_set_num_threads(int num)`
- `int omp_get_max_threads(void)`
- `int omp_get_num_threads(void)`
- `int omp_get_thread_num (void)`
- `int omp_get_num_procs (void)`
- `int omp_in_parallel (void)`
- `void omp_set_dynamic(int num)`
- `int omp_get_dynamic(void)`
- `void omp_get_nested(void)`
- `void omp_set_nested(int nested)`

Библиотека функций OpenMP

- `void omp_init_lock(omp_lock_t *lock)`
- `void omp_nest_init_lock(omp_nest_lock_t *lock)`
- `void omp_destroy_lock(omp_lock_t *lock)`
- `void omp_destroy_nest_lock(omp_nest_lock_t *lock)`
- `void omp_set_lock(omp_lock_t *lock)`
- `void omp_set_nest_lock(omp_nest_lock_t *lock)`
- `void omp_unset_lock(omp_lock_t *lock)`
- `void omp_unset_nest_lock(omp_nest_lock_t *lock)`
- `void omp_test_lock(omp_lock_t *lock)`
- `void omp_test_nest_lock(omp_nest_lock_t *lock)`

Переменные среды OpenMP

- OMP_SCHEDULE
- OMP_NUM_THREADS
- OMP_DYNAMIC
- OMP_NESTED

Переменные среды OpenMP

```
#include <omp.h>
THREADNUMS 2
|
int main(int argc, char *argv[]) {
    Long StepNums = 10000;
    double step, x, pi, sum=0.0;
    int i;
    step = 1.0/StepNums;
    omp_st_threads(THREADNUMS);
    #pragma omp parallel for reduction (+:sum) private (l,x)
    for (i=0; i<StepNums; i++) {
        x=(i-0.5)*step;
        sum = sum + 4.0/(1.0 - x*x);
    }
    pi = step*sum;
}
```