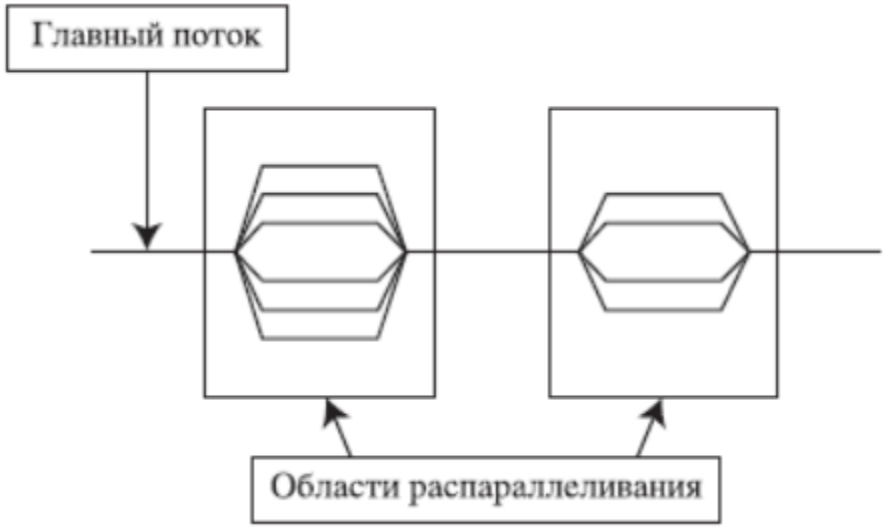
# OpenMP




Open Multi-Processing

Директивы компилятора, библиотечные процедуры и переменные окружения

Системы с общей памятью

Языки: Си, Си++ и Фортран

#pragma omp конструкция [предложение [предложение] … ]

# OpenMP

```
#include <omp.h> // Open Multi-Processing


int main(int argc, char **argv) {
	omp_set_num_threads(N); // установить число потоков в N
#pragma omp parallel // директива компилятору
{
	// параллельное исполнение
}
	return 0x00;
}
```

# OpenMP

Компиляция:

```
gcc my_openMP_prog.c -o my_openMP_prog -fopenmp
```

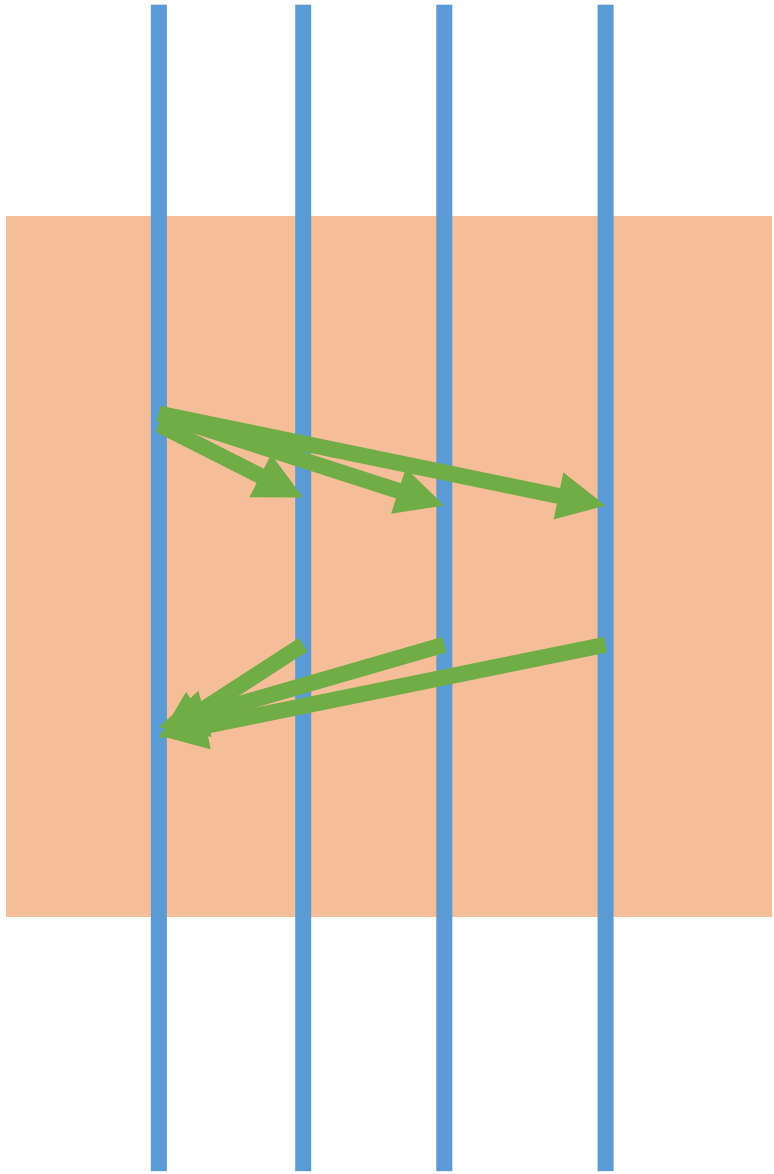
# OpenMP

```
#pragma omp parallel
{
    // parallel указывает, что данный блок кода должен быть исполнен
    // параллельно в несколько потоков
}

#pragma omp parallel // сокращенная запись
    … // блок кода, исполняющийся параллельно
```

# MPI

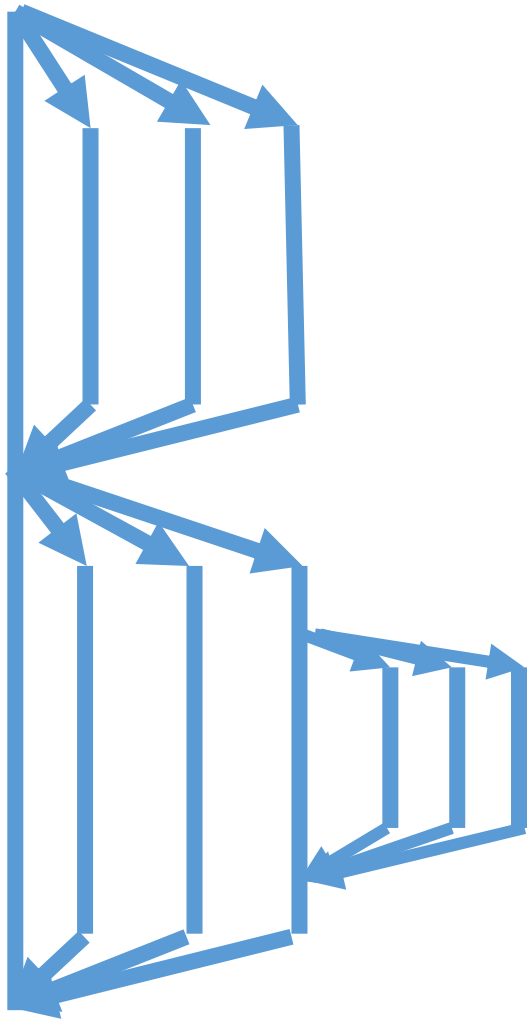
```
MPI_Init(...);
...
MPI_Scatter(...);
...
MPI_Gather(...);
...
MPI_Finalize();
```



# OpenMP

```
#pragma omp parallel num_threads(4)
{
    ...
    ...
    ...
}

#pragma omp parallel num_threads(4) sections
{
    ...
    #pragma omp section
    {
        #pragma omp parallel num_threads(4)
        {
            ...
        }
    }
    ...
}
```



# OpenMP

```
#pragma omp parallel
{
    #pragma omp for
    for (int  i = 0; i < K; i++) // параллельное суммирование чисел 0..K
        res += i; // с теоретическим ускорением N, где N - число потоков
}

#pragma omp parallel for // сокращенная запись
    … // цикл for, исполняющийся параллельно
```

Код содержит состояние гонки

# OpenMP

```
#pragma omp parallel shared(a) private(myid, x)
{
    myid = omp_get_thread_num();
    x = work(myid);
    if(x < 1.0)
        a[myid] = x;
}
#pragma omp parallel default(private) shared(a)
{
    myid = omp_get_thread_num();
    x = work(myid);
    if(x < 1.0)
        a[myid] = x;
}
```

# OpenMP

firstprivate(var1, var2, ...) private + указанные переменные инициализируются значением до входа в параллельную секцию.

```
int myid;
int a = 10;
#pragma omp parallel default(private) \
 firstprivate(a)
{
    myid = omp_get_thread_num();
    printf("Thread%d: a = %d\n", myid, a);
    a = myid;
    printf("Thread%d: a = %d\n", myid, a);
}
```

# OpenMP

lastprivate(var1, var2, ...) Приватные переменные сохраняют свое значение, которое они получили при достижении конца параллельного участка кода.

```
#pragma omp parallel
{
    #pragma omp for private(i) lastprivate(k)
    for(i=0; i<10; i++)
        k = i*i;
}
printf("k = %d\n", k);
```

# OpenMP

sections / section – разделение задач между потоками

```
#pragma omp parallel sections ///nowait
{
    #pragma omp section
    {
        printf("T%d: task1\n", omp_get_thread_num());
    }
    #pragma omp section
    {
        printf("T%d: task1\n", omp_get_thread_num());
    }
}
```

# OpenMP

reduction(оператор:var1, var2, ...) гарантирует безопасное выполнение операций редукции, например, вычисление глобальной суммы.

```
#pragma omp parallel
{
    #pragma for shared(x) private(i) reduction(+:sum)
    for(i=0; i<10000; i++)
        sum += x[i];
}
#pragma omp parallel
{
    #pragma for shared(x) private(i) reduction(min:gsum)
    for(i=0; i<10000; i++)
        gmin = min(gmin, x[i]);
}
```

+, - , *, &, ^, |, &&, ||, min, max

# OpenMP

if(выражение) параллельное выполнение необходимо только если выражение истинно.

```
#pragma omp parallel
{
    #pragma omp for if(n>2000)
    {
        for(i=0; i<n; i++)
            a[i] = work(i);
    }
}
```

# OpenMP pragmas

- num_threads(numThreads) – установка количества тредов
- sections / section – разделение задач между потоками
- single – при необходимости сделать действие одним потоком в параллельном участке
- critical – критическая секция
- atomic – атомарность операции
- barrier – точка синхронизации
- if(выражение) параллельное выполнение необходимо только если выражение истинно.

# OpenMP pragmas with for

- private(var1, var2, ...) переменные создают локальные копии каждому потоку для избегания состояния гонки.
- firstprivate(var1, var2, ...) private + указанные переменные инициализируются значением до входа в параллельную секцию.
- lastprivate(var1, var2, ...) Приватные переменные сохраняют свое значение, которое они получили при достижении конца параллельного участка кода.
- reduction(оператор:var1, var2, ...) гарантирует безопасное выполнение операций редукции, например, вычисление глобальной суммы.
- ordered – в параллельных циклах говорит о исполнении в строго фиксированной последовательности

# OpenMP pragmas with for

schedule(тип [, размер блока])

static – итерации равномерно распределяются по потокам, нудным размером блока.

dynamic – работа распределяется пакетами заданного размера между потоками. При завершении текущего блока берёт следующий.

guided – dynamic + Размер блока постепенно уменьшается вплоть до указанного значения.

# OpenMP Functions

omp_get_thread_num() – номер текущего потока

omp_get_num_threads() – общее количество потоков

omp_get_num_procs() – количество физических процессоров

omp_set_num_threads(int num_threads) – установить количество потоков

omp_set_nested(int nested) – разрешить/запретить вложенный параллелизм

omp_in_parallel() – возвращает не нулевое значение в параллельном блоке, иначе 0

# OpenMP ASM

```
1   parallelTest():
2           push    rbp
3           mov     rbp, rsp
4           call    omp_get_thread_num
5           mov     edi, eax
6           call    sleep
7           nop
8           pop     rbp
9           ret
```

```
#include <unistd.h>
#include <omp.h>
void parallelTest() {
    #pragma omp parallel num_threads(3)
    {
        sleep(omp_get_thread_num());
    }
}
```

```
1   parallelTest():
2           push    rbp
3           mov     rbp, rsp
4           mov     ecx, 0
5           mov     edx, 3
6           mov     esi, 0
7           mov     edi, OFFSET FLAT:parallelTest() [clone ._omp_fn.0]
8           call    GOMP_parallel
9           nop
10          pop     rbp
11          ret
12  parallelTest() [clone ._omp_fn.0]:
13          push    rbp
14          mov     rbp, rsp
15          sub     rsp, 16
16          mov     QWORD PTR [rbp-8], rdi
17          call    omp_get_thread_num
18          mov     edi, eax
19          call    sleep
20          leave
21          ret
```

# OpenMP Links

- Официальная документация: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf
- Подробный учебник: https://parallel.ru/sites/default/files/info/parallel/openmp/OpenMP.pdf
- Параллельные заметки №1-5