Методы обеспечения совместимости (для языка С)

- Если компилятор не поддерживает openmp, директивы #pragma omp игнорируются
- Макрос _OPENMP (показывает наличие поддержки OPENMP и версию в формате yyyymm, где yyyy – год, а mm – месяц принятия соответствующей версии стандарта
- Директива if

Примеры кода

```
#ifdef _OPENMP
    int max_threads = omp_get_max_threads();
//(201307 – версия openmp 4.0)
#if _OPENMP >= 201307
#pragma omp simd
    for (int i = 0; i < 16; ++i)  //simd рассмотрим позже
          data[i] += x[i];
#else
     for (int i = 0; i < 16; ++i)
           data[i] += x[i];
#endif
```

Примеры кода - 2

```
#include <omp.h>
#include <stdio.h>
int main (void) {
    int t = 1;
    int f = 0;
#pragma omp parallel if (f)
        printf ("FALSE: I am thread %d\n", omp_get_thread_num());
 #pragma omp parallel if (t)
        printf ("TRUE : I am thread %d\n", omp_get_thread_num());
    return 0;
}
```

********** запуск на 4 потоках

```
FALSE: I am thread 0
TRUE : I am thread 0
TRUE : I am thread 1
TRUE : I am thread 3
TRUE : I am thread 2
```

Число потоков

- Глобальная переменная OMP_NUM_THREADS

(bash# export OMP_NUM_THREADS=16)

- Функция void omp_set_num_threads(int num)

  Задаёт число потоков в предстоящих параллельных областях, т.е. по сути переопределяет глобальную переменную в рамках программы, пример вызова

  omp_set_num_threads(16);

- Директива num_threads(num)

  Задаёт число потоков соответствующей параллельной области

  #pragma omp parallel num_threads(8)

  {что-то, что выполнится при помощи 8-ми потоков}

Число потоков – 2

- int omp_get_max_threads(void)

  сколько всего можно создать потоков

- int omp_get_num_procs(void)

  сколько всего процессорных ядер?

  //чаще всего для вычислительных задач больше потоков создавать нет смысла)

- int omp_get_num_threads(void)

  сколько всего потоков в параллельной области

- int omp_get_thread_num(void)

  узнать свой номер – можно использовать для построения алгоритмов

Число потоков - дополнительно

- Управление динамической настройкой числа исполнителей в процессе выполнения программы - переменная среды OMP_DYNAMIC, функции
  void omp_set_dynamic(int num) и
  int omp_get_dynamic(void)
- Узнать, в параллельной ли мы области? Функция
  int omp_in_parallel(void)
- Вложенный параллелизм и его максимальный уровень – переменные среды OMP_NESTED и OMP_MAX_ACTIVE_LEVELS, функции
  void omp_set_nested(int nested) ,
  int omp_get_nested(void),
  int omp_get_team_size(int level) и другие

Программа создана. Измерим производительность?

- double omp_get_wtime(void)

  //имеет смысл только разница между двумя значениями, абсолютное значение зависит от реализации, значения в разных потоках не синхронизированы
- double omp_get_wtick(void)

  узнаём разрешение таймера

Распараллеливание циклов

- Директива for

```
#pragma omp for
    for (i = 0; i < 128; i++)
```

(внутри параллельной области)

```
#pragma omp parallel for
    for (i = 0; i < 128; i++)
//распараллеливается по индексу, цикл проходится один раз
//нет проверки информационных зависимостей!
//без директивы for будет полная неразбериха, так как
  переменная i – общая, если объявить её в теле цикла –
  цикл выполнится столько раз, сколько есть потоков
```

Требования к циклу for

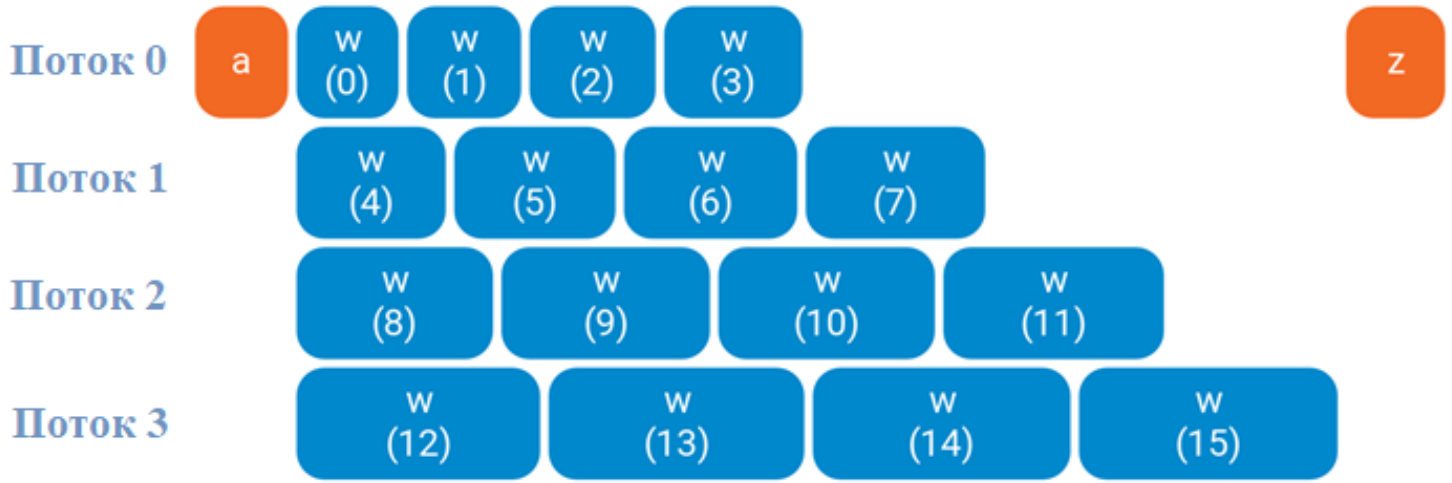
- Зависят от версии стандарта (чем старее стандарт, тем жёстче)
- Не должно быть дополнительных точек выхода вроде break
- Во всех версиях работает целочисленным итератором, фиксированным инкрементом/декрементом, заранее определённым значением интервала

Как управлять распараллеливанием?

- Есть возможность управления балансировкой (**scheduling**)

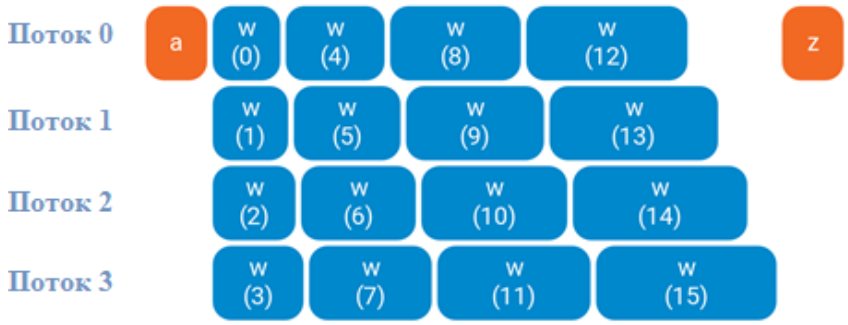
Без директивы

```
a();
#pragma omp parallel for
    for (int i = 0; i < 16; ++i)
        { w(i); }
z();
```

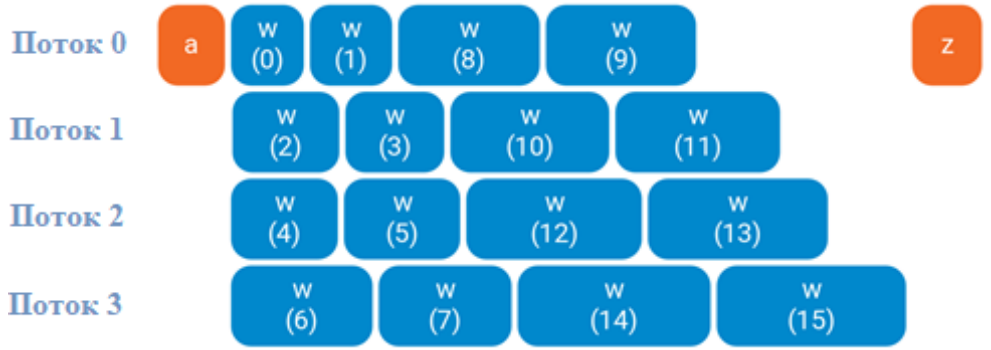


Директива (static, chunk – не обязателен)

```
a();
#pragma omp parallel for schedule(static,1)
   for (int i = 0; i < 16; ++i)
      { w(i); }
z();
```

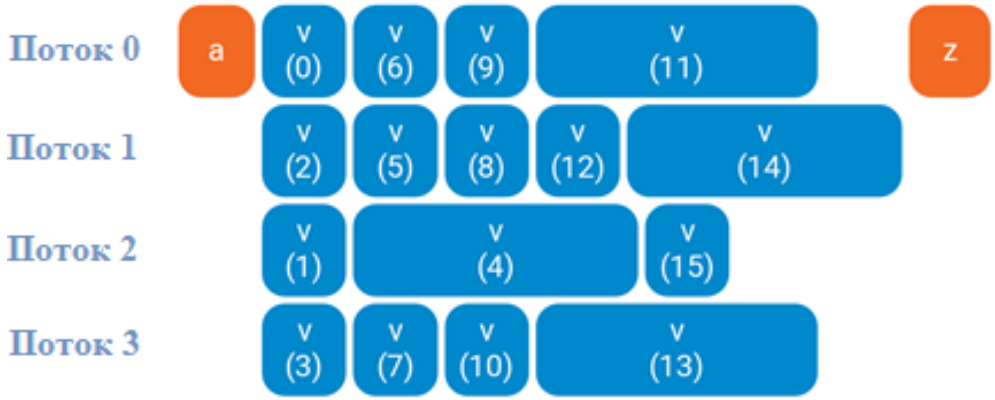



```
a();
#pragma omp parallel for schedule(static,2)
   for (int i = 0; i < 16; ++i)
      { w(i); }
z();
```



Директива (dynamic , chunk – не обязателен)

```
#pragma omp parallel for schedule(dynamic)
   for(int i=0; i<16; i++) printf(" %d \n", i);
// конвеерный параллелизм, по 1 итерации, аналогично
a();
#pragma omp parallel for schedule(dynamic,1)
        for (int i = 0; i < 16; ++i)
            {v(i); }
z();
```




```
#pragma omp parallel for schedule(dynamic,4)
   for(int i=0; i<16; i++) printf(" %d \n", i);
// конвеерный параллелизм, по 4 итерации
```

Директивы guided, auto

- guided – используем преимущества dynamic и снижаем накладные расходы за счёт динамически меняющегося шага (от большего к меньшему)
- auto – на выбор системы

//анонс первой задачи со * - на 0.3 балла

//распараллелить цикл на 65 итераций на 4 потоках разными методами балансировки и объяснить результаты

Зоны видимости переменных

- По умолчанию – всё что вне блока – общие, что внутри блока – приватные (т.е. разные для потоков)

```
#pragma omp parallel
 {
   int num;
   num = omp_get_thread_num();
   printf(“%d\n”,num);// у каждого потока переменная num своя
}
```

- Можно управлять директивами

Зоны видимости переменных - 2

- private (список переменных)//через запятую

(переменные создаются, но не инициализируются)

- shared (список переменных)
- firstprivate (список переменных) // тут вновь создаваемые переменные инициализируются значением до параллельной области
- lastprivate (список переменных) // после выполнения параллельной области в переменную будет записано последнее значение – из последней итерации цикла или из последней секции

Зоны видимости переменных - 3

- default //shared или none, в последнем случае придётся прописывать все переменные в других директивах
- reduction (операция:список переменных)

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

(примечание: - работает также как +)

Для аддитивных – новые копии переменных устанавливаются в 0. Для мультипликативных – в 1.

- threadprivate(список переменных) (для мастер-потока старая переменная, остальным – новая, не инициализированная)
- copyin(список переменных) //разобраться самостоятельно и написать пример – на 0.2 балла со *

Особняком – итераторы циклов