Введение в параллельные вычисления

д.т.н. Мокрова Наталия Владиславовна

асп. Морунов Егор, Сырко Денис

пятница	ауд. 118, 119	
12:50 – 14:20	Лекция	
	1 неделя	2 неделя
14:35 - 17:55	Лаб – КС44	Лаб - КС40

Выбор модели программирования

Системы с общей памятью

- C++11 threads (pthreads)
- OpenMP

Системы с распределенной памятью

- MPI = message passing interface

Гетерогенные системы (графические процессоры)

- CUDA

Лекция 6. Технология OpenMP. Основные конструкции

The OpenMP® API specification for parallel programming
http://openmp.org

Антонов А.С. Параллельное программирование с использованием технологии OpenMP. М.: Издво МГУ, 2009.

Примеры из учебника «Технологии параллельного программирования MPI и OpenMP»
http://parallel.ru/tech/tech_dev/MPI%26OpenMP/examples

Примеры http://pro-prof.com/archives/1150

Демонстрация примеров http://coliru.stacked-crooked.com/

Пример

Найти ошибку, чтобы вывод мог быть в виде:

```
Thread:0Thread:
Master:4
Thread:3
Thread:2
1
```

omp_get_thread_num() – определение номера нити в текущей параллельной секции;
omp_get_num_threads() – количество нитей.
По умолчанию количество потоков равно количеству логических процессоров в системе.

```
#include <iostream>
#include <omp.h>
using namespace std;
int main()
{
 int nthreads, tid;
#pragma omp parallel
 {
  tid = omp_get_thread_num();
  cout << "Thread:" << tid << endl;
  if (tid == 0) {
   nthreads = omp_get_num_threads();
   cout << "Master:" << nthreads << endl;
  }
 }
 return 0;
}
```

Директивы и функции

Директива компилятору – выполнять программу в параллельном режиме.

Формат директивы на Си/Си++:

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

Объектом действия большинства директив является один оператор или блок, перед которым расположена директива, такие блоки – ассоциированные с директивой.

Ассоциированный блок имеет одну точку входа в начале и одну точку выхода в конце, порядок опций в описании директивы несущественен, в одной директиве большинство опций может встречаться несколько раз, после некоторых опций может следовать список переменных.

Директивы OpenMP можно разделить на категории:
- определение параллельной области,
- распределение работы,
- синхронизация.

Каждая директива может иметь несколько дополнительных атрибутов – опций (clause). Отдельно специфицируются опции для назначения классов переменных, которые могут быть атрибутами различных директив.

Выполнение программы

Функции библиотеки OpenMP – подключить заголовочный файл omp.h

Запустить выполняемый файл на требуемом количестве процессоров, т.е. задать количество нитей, можно определив значение переменной среды OMP_NUM_THREADS.

export OMP_NUM_THREADS=n – для Linux в bash.

void omp_set_num_threads(int num_threads) – задать количество потоков из программы.

Замер времени

Функция для работы с системным таймером omp_get_wtime() возвращает в вызвавшей нити астрономическое время в секундах (вещественное число двойной точности), прошедшее с некоторого момента в прошлом.

double omp_get_wtime(void);

Момент времени, используемый в качестве точки отсчета, не будет изменён за время существования процесса. Таймеры разных нитей могут быть не синхронизированы и выдавать различные значения.

Функция **omp_get_wtick()** возвращает в вызвавшей нити разрешение таймера в секундах. Это время можно рассматривать как меру точности таймера.

время в секундах с некоторого момента в прошлом

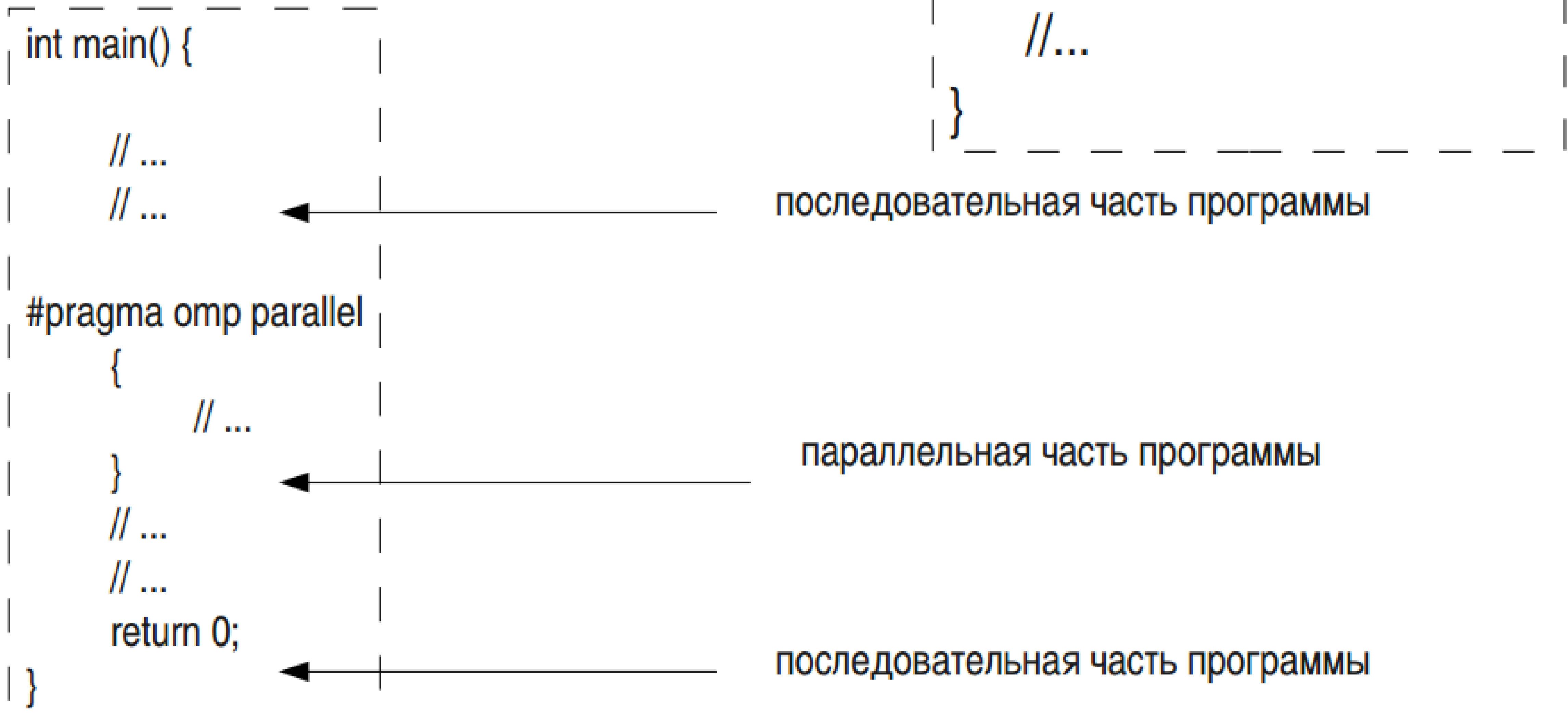
разрешение таймера в секундах

```
//пример
double start = omp_get_wtime();

//...
//...

double end = omp_get_wtime();
cout<<end-start<<endl;
```

Директива parallel



Опции директивы parallel

Опция	Значение
if (условие)	выполнение параллельной области по условию (если условие не выполнено, то директива не срабатывает);
num_threads (целочисленное выражение)	явное задание количества нитей; по умолчанию выбирается последнее значение, установленное omp_set_num_threads();
default (shared | none)	всем переменным в параллельной области, будет назначен класс (none – класс должен быть назначен явно);
private (список)	задаёт список переменных, для которых порождается локальная копия в каждой нити (начальное значение не определено);
firstprivate (список)	тоже порождается локальная копия в каждой нити, которые инициализируются значениями в нити-мастере;
shared (список)	задаёт список переменных, общих для всех нитей;
copyin (список)	копирует значения переменных threadprivate из главного потока в переменные threadprivate дочерних;
reduction (оператор: список)	по окончании параллельного блока применить оператор к переменным из списка (собрать результаты в гл. потоке).

Детализация опции reduction(оператор : список)

- локальные копии инициализируются соответственно типу оператора (для аддитивных операций – 0 или его аналоги, для мультипликативных операций – 1 или её аналоги);
- над локальными копиями переменных после выполнения всех операторов параллельной области выполняется заданный оператор;
- оператор это: для языка Си – +, *, -, &, |, ^, &&, ||;
- порядок выполнения операторов не определён, поэтому результат может отличаться от запуска к запуску.

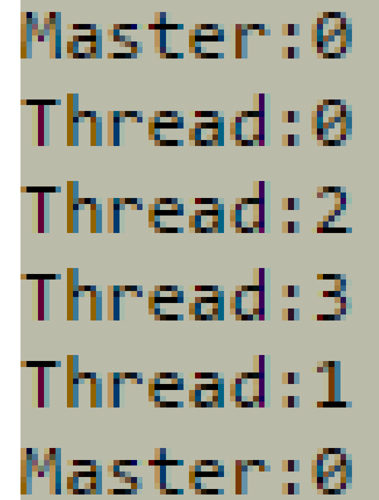
Особенности реализации

При входе в параллельную область порождаются OMP_NUM_THREADS-1 нитей, каждая нить получает свой уникальный номер, причём порождающая нить получает номер 0 и становится основной нитью группы («мастером»). Количество нитей, выполняющих данную параллельную область, остаётся неизменным до момента выхода из области. При выходе из параллельной области производится неявная синхронизация и уничтожаются все нити, кроме породившей. Все порождённые нити исполняют один и тот же код, соответствующий параллельной области. Предполагается, что в SMP-системе нити будут распределены по различным процессорам (однако это, как правило, находится в ведении операционной системы).

Параллельные и последовательные области

Директива parallel (см. Логические ошибки, отсутствие parallel)

```
#include <iostream>
#include <omp.h>
using namespace std;
int main()
{
   cout << "Master:" << omp_get_thread_num() << endl;
#pragma omp parallel
       {
                 cout << "Thread:" << omp_get_thread_num()<< endl;
       }
    cout << "Master:" << omp_get_thread_num() << endl;
  }
```



Параллельные и последовательные области

```
int main()
{
cout << "Master:" << omp_get_thread_num() << endl;
#pragma omp parallel for
for (int i = 0; i < omp_get_num_threads(); i++)
        cout << "Thread:" << omp_get_thread_num()<< endl;
cout << "Master:" << omp_get_thread_num() << endl;
}
```

Вывод одинаковый для 3-х фрагментов

```
int main()
{
    cout << "Master:" << omp_get_thread_num() << endl;
#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < omp_get_num_threads(); i++)
            cout << "Thread:" << omp_get_thread_num()<< endl;
}
    cout << "Master:" << omp_get_thread_num() << endl;
}
```

Пример

Какой вывод?

```
int main()
{
    cout << "Master:" << omp_get_thread_num() << endl;
#pragma omp parallel
{
    #pragma parallel for
    for (int i = 0; i < omp_get_num_threads(); i++)
            cout << "Thread:" << omp_get_thread_num()<< endl;
}
    cout << "Master:" << omp_get_thread_num() << endl;
}
```

Применение опции reduction

Производится подсчет общего количества порождённых нитей.

Каждая нить инициализирует локальную копию переменной count значением 0.

Каждая нить увеличивает значение собственной копии переменной count на единицу и выводит полученное число.

На выходе из параллельной области – суммирование значений переменных count по всем нитям.

Полученная величина – новое значение переменной count в последовательной области.

```
    int count = 0;
#pragma omp parallel reduction (+: count)
    {
        count++;
          cout << "Текущее значение count" << count << endl;
    }
    cout << "Число нитей: " << count;
```

С и С++

```c
#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);
}
```

```
Текущее значение count: 1
Текущее значение count: 1
Текущее значение count: 1
Текущее значение count: 1
Число нитей: 4
```

```cpp
#include <iostream>
#include <stdio.h>
using namespace std;
int main()
{
    int count = 0;
    #pragma omp parallel reduction (+: count)
    {
        count++;
        cout << "Текущее значение count"  << count << endl;
    }
cout << "Число нитей: " << count;
}
```

Пример: опций if, num_threads

```
I am thread № 0
```

```
1  #include <iostream>
2  #include <omp.h>
3  using namespace std;
4  int main()
5  {
6   int is_parallel=0;
7   #pragma omp parallel if(is_parallel == 1) num_threads(8)
8   {
9      cout<<"I am thread № "<< omp_get_thread_num() << endl;
10  }
11 }
```

или

```
if(is_parallel == 0) num_threads(8)
```

Пример: опция if

if используется для организации условного выполнения потоков в параллельном структурном блоке.

В примере цикл распараллеливается только в том случае (n > 2000), когда параллельная версия заведомо быстрее последовательной (трудоемкость образования параллельных потоков эквивалентна примерно трудоемкости 1000 операций). *Иначе выполняется последовательно.*

Левин М.П. Параллельное программирование с использованием OpenMP: учебное пособие. – М: Интернет-Университет Информационных технологий; Бином, 2014. – 118 с.

```
1  #include <iostream>
2  #include <omp.h>
3  #include <climits>
4  using namespace std;
5  const int n = 100;
6  int a[n];
7  int main()
8  { int s;
9      for (int i=0; i<n; i++) {
10         a[i] = rand() % 20 + 1;
11         cout << a[i] << " " ;
12     }
13 #pragma omp parallel if( n > 2000 )
14  {
15     for (int i=0; i<n; i++) {
16          s = s + a[i];
17     }
18  }
19  cout << " S = " << s << endl;
20 }
```

Переменные среды и вспомогательные функции

Перед первой параллельной областью вызовом функции **omp_set_num_threads(2)** выставляется количество нитей, равное 2.

К первой параллельной области применяется опция **num_threads(3)**, данную область следует выполнять тремя нитями.

Сообщение "Параллельная область 1" выведено тремя нитями.

Ко второй параллельной области опция **num_threads** не применяется, поэтому действует значение, установленное функцией **omp_set_num_threads(2)**, и сообщение "Параллельная область 2" выведено двумя нитями.

```
1  #include <iostream>
2  #include <omp.h>
3  using namespace std;
4  int main()
5  {
6      omp_set_num_threads(2);
7      #pragma omp parallel num_threads(3)
8      {
9          cout << "Параллельная область 1" << endl;
10     }
11     #pragma omp parallel
12     {
13         cout << "Параллельная область 2" << endl;
14     }
15 }
```

```
Параллельная область 1
Параллельная область 1
Параллельная область 1
Параллельная область 2
Параллельная область 2
```

Динамическое изменение количества нитей

Для оптимизации использования ресурсов системы.

Необходимо переменную среды **OMP_DYNAMIC** установить в true.
Команда:
export OMP_DYNAMIC = true
в bash или функция:
omp_set_dynamic().

По умолчанию: false.

Узнать значение переменной OMP_DYNAMIC можно при помощи функции omp_get_dynamic().

```
1  #include <iostream>
2  #include <omp.h>
3  using namespace std;
4  int main()
5  {
6   cout << "Значение OMP_DYNAMIC " << omp_get_dynamic() << endl;
7   omp_set_dynamic(1);
8   cout << "Значение OMP_DYNAMIC " << omp_get_dynamic() << endl;
9  #pragma omp parallel num_threads(128)
10  {
11 #pragma omp master
12  {
13  cout << "Параллельная область нитей: " << omp_get_num_threads();
14  }
15  }
16 }
```

```
Значение OMP_DYNAMIC 0
Значение OMP_DYNAMIC 1
Параллельная область нитей: 4
```

Вложенные параллельные области

Функция **omp_get_max_threads()** возвращает максимально допустимое число нитей для использования в следующей параллельной области.

int omp_get_max_threads(void);

Функция **omp_get_num_procs()** возвращает количество процессоров, доступных для использования программе на момент вызова. Нужно учитывать, что количество доступных процессоров может динамически изменяться.

int omp_get_num_procs(void);

Параллельные области могут быть вложенными; по умолчанию вложенная параллельная область выполняется одной нитью. Это управляется установкой переменной среды OMP_NESTED. Разрешить вложенный параллелизм можно при помощи команды: export OMP_NESTED = true (в bash). Изменить значение OMP_NESTED – функция **omp_set_nested()**.

void omp_set_nested(int nested)

Функция **omp_set_nested()** разрешает или запрещает вложенный параллелизм. На языке Си в качестве значения параметра задаётся 0 или 1. Если вложенный параллелизм разрешён, то каждая нить, в которой встретится описание параллельной области, породит для её выполнения новую группу нитей. Сама породившая нить станет в новой группе нитью-мастером. Если система не поддерживает вложенный параллелизм, данная функция не будет иметь эффекта.

Использование вложенных параллельных областей

Вызов функции **omp_set_nested()** перед первой частью разрешает использование вложенных параллельных областей.

Для определения номера нити в текущей параллельной секции используются вызовы функции **omp_get_thread_num()**.

Каждая нить внешней параллельной области породит новые нити, каждая из которых напечатает свой номер вместе с номером породившей нити.

Вызов **omp_set_nested()** запрещает использование вложенных параллельных областей.

Во второй части вложенная параллельная область будет выполняться без порождения новых нитей (см. листинг).

https://hpc.icc.ru/documentation/msu/OpenMP.pdf

```
Часть 1, нить 0 - 0
Часть 1, нить 1 - 1
Часть 1, нить 0 - 1
Часть 1, нить 1 - 2
Часть 1, нить 1 - 0
Часть 1, нить 0 - 3
Часть 1, нить 3 - 0
Часть 1, нить 3 - 2
Часть 1, нить 1 - 3
Часть 1, нить 0 - 2
Часть 1, нить 2 - 1
Часть 1, нить 3 - 1
Часть 1, нить 2 - 2
Часть 1, нить 2 - 3
Часть 1, нить 2 - 0
Часть 1, нить 3 - 3
Часть 2, нить 3 - 0
Часть 2, нить 1 - 0
Часть 2, нить 0 - 0
Часть 2, нить 2 - 0
```

Вложенные параллельные области

Узнать значение переменной OMP_NESTED можно при помощи функции **omp_get_nested()**.

int omp_get_nested(void);

Функция **omp_in_parallel()** возвращает 1, если находимся в параллельной секции

int omp_in_parallel(void);

Пример иллюстрирует применение функции **omp_in_parallel()**. Функция mode демонстрирует изменение функциональности в зависимости от того, вызвана она из последовательной или из параллельной области.

```
1  #include <iostream>
2  #include <omp.h>
3  using namespace std;
4  void mode(void){
5   if(omp_in_parallel()) cout << "Параллельная область" ;
6   else cout << "Последовательная область" << endl;
7  }
8  int main()
9  {
10  mode();
11 #pragma omp parallel
12  {
13 #pragma omp master
14  {
15  mode();
16  }
17  }
18 }
```

```
Последовательная область
Параллельная область
```

Директива single

Если в параллельной области какой-либо участок кода должен быть выполнен лишь один раз, то его нужно выделить директивами **single**.

#pragma omp single [опция [[,] опция]...]

Возможные опции:

private(список) – задаёт список переменных, для которых порождается локальная копия в каждой нити; начальное значение локальных копий переменных из списка не определено;

firstprivate(список) – задаёт список переменных, для которых порождается локальная копия в каждой нити; локальные копии переменных инициализируются значениями этих переменных в нити-мастере;

copyprivate(список) – после выполнения нити, содержащей конструкцию single, новые значения переменных списка будут доступны всем одноименным частным переменным (private и firstprivate), описанным в начале параллельной области и используемым всеми её нитями; опция не может использоваться совместно с опцией nowait; переменные списка не должны быть перечислены в опциях private и firstprivate данной директивы single;

nowait – после выполнения выделенного участка происходит неявная барьерная синхронизация параллельно работающих нитей: их дальнейшее выполнение происходит только тогда, когда все они достигнут данной точки; если в подобной задержке нет необходимости, опция nowait позволяет нитям, уже дошедшим до конца участка, продолжить выполнение без синхронизации с остальными.

Пример

Какая именно нить будет выполнять выделенный участок программы, не специфицируется. Одна нить будет выполнять данный фрагмент, а все остальные нити будут ожидать завершения её работы, если только не указана опция nowait.

Необходимость single часто возникает при работе с общими переменными.

Пример иллюстрирует применение директивы single вместе с опцией nowait.

Сначала все нити напечатают текст "Сообщение 1", при этом одна нить (не обязательно нить-мастер) дополнительно напечатает текст "Одна нить". Остальные нити, не дожидаясь завершения выполнения области single, напечатают текст "Сообщение 2". Таким образом, первое появление "Сообщение 2" в выводе может встретиться как до текста "Одна нить", так и после него. Если убрать опцию nowait, то по окончании области single произойдёт барьерная синхронизация, и ни одна выдача "Сообщение 2" не может появиться до выдачи "Одна нить".

```
1  #include <iostream>
2  #include <omp.h>
3  using namespace std;
4  #include <stdio.h>
5  int main()
6  {
7  #pragma omp parallel
8   {
9   cout << "  Сообщение 1 " << endl;
10 #pragma omp single nowait
11  {
12  cout << " Одна нить " << endl;
13  }
14  cout << "Сообщение 2 " << endl;
15  }
16 }
```

```
  Сообщение 1

 Одна нить
Сообщение 2
  Сообщение 1
Сообщение 2
  Сообщение 1
Сообщение 2
Сообщение 2
```

Применение опции copyprivate

В данном примере переменная n объявлена в параллельной области как локальная.

Каждая нить присвоит переменной n значение, равное своему порядковому номеру, и напечатает данное значение. В области single одна из нитей присвоит переменной n значение 100, и на выходе из области это значение будет присвоено переменной n на всех нитях.

В конце параллельной области значение n печатается ещё раз и на всех нитях оно равно 100.

```
1  #include <iostream>
2  #include <omp.h>
3  int main(int argc, char *argv[])
4  {
5      int n;
6      #pragma omp parallel private(n)
7      {
8          n =omp_get_thread_num();
9          printf("Значение n (начало): %d\n", n);
10         #pragma omp single copyprivate(n)
11         {
12                 n=100;
13         }
14         printf("Значение n (конец): %d\n", n);
15     }
16 }
```

```
Значение n (начало): 0
Значение n (начало): 3
Значение n (начало): 2
Значение n (начало): 1
Значение n (конец): 100
Значение n (конец): 100
Значение n (конец): 100
Значение n (конец): 100
```

Задания

- Откомпилируйте последовательную программу с включением опций поддержки технологии OpenMP и запустите с использованием нескольких нитей. Сколько нитей будет реально исполнять операторы данной программы?
- Может ли программа на OpenMP состоять только из параллельных областей? Только из последовательных областей?
- Чем отличается нить-мастер от всех остальных нитей?
- При помощи функций OpenMP определите время, при помощи функции omp_get_wtick(). Хватает ли для этого точности системного таймера?