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 7. Технологии OpenMP. Распределение работы

Параллельные и последовательные области (см. Лекция 6).

- Директива master.

Модель данных.

Распределение работы.

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/

Директива master

Директива master выделяет участок кода, который будет выполнен только нитью-мастером.

Остальные нити пропускают данный участок и продолжают работу с оператора, расположенного далее. Неявной синхронизации данная директива не предполагает.

#pragma omp master

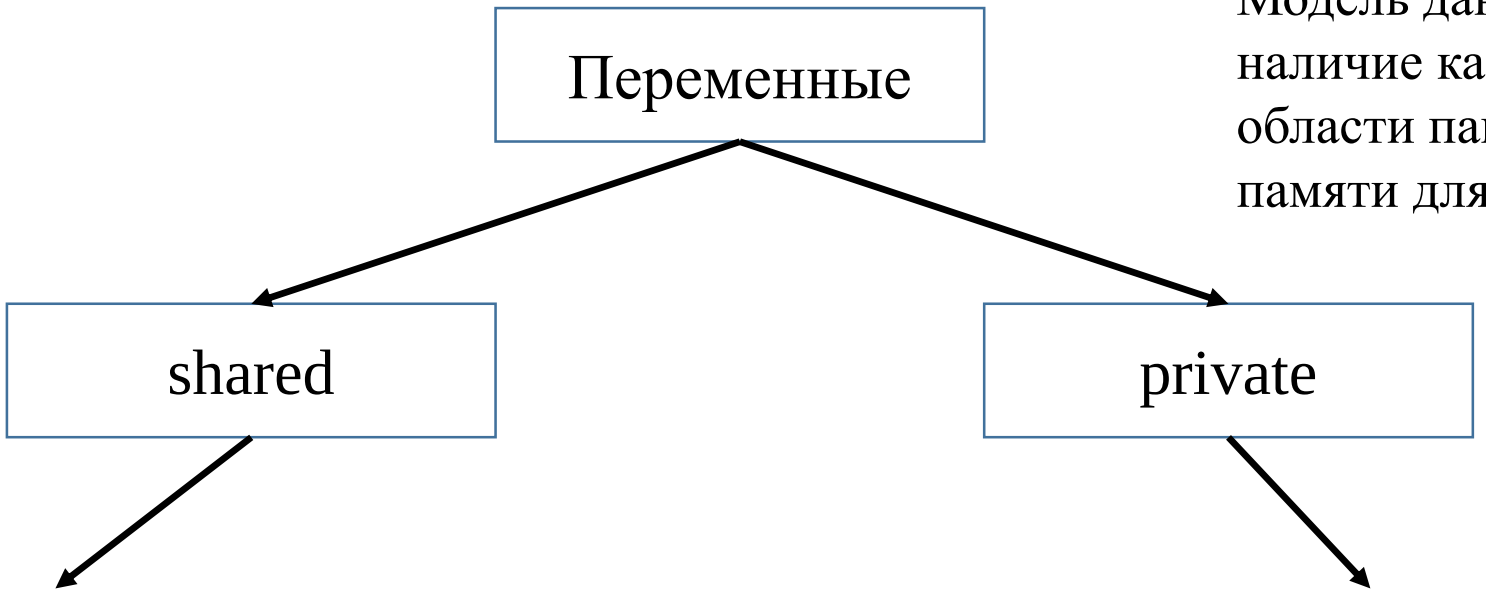
Переменная n – локальная, каждая нить работает со своим экземпляром.
Все нити n = 1.
Нить-мастер присвоит переменной n значение 2, и все нити напечатают значение n.
Затем нить-мастер присвоит переменной n значение 3, и снова все нити напечатают значение n.
Директиву master всегда выполняет одна и та же нить.

```
Первое значение n: 2
Первое значение n: 1
Первое значение n: 1
Первое значение n: 1
Второе значение n: 1
Второе значение n: 1
Второе значение n: 3
Второе значение n: 1
```

```
 5  int main(int argc, char *argv[])
 6  {
 7     int n;
 8  #pragma omp parallel private(n)
 9     {
10        n=1;
11  #pragma omp master
12        {
13           n=2;
14        }
15        cout << "Первое значение " << n << endl;
16  #pragma omp barrier
17  #pragma omp master
18        {
19           n=3;
20        }
21        cout << "Второе значение " << n << endl;
22     }
23  }
```

Модель данных в OpenMP

Модель данных в OpenMP предполагает наличие как общей для всех нитей области памяти, так и локальной области памяти для каждой нити.



общие для всех нитей.

локальные, для каждой нити своя переменная (типы и имена могут совпадать);
не определены до и после блока параллельных вычислений.

По умолчанию:

- переменные объявленные вне параллельной области – shared;
- переменные, объявленные в параллельной области – private.

Классы shared и private

Общая переменная всегда существует лишь в одном экземпляре для всей области действия и доступна всем нитям под одним и тем же именем.

Объявление локальной переменной вызывает порождение своего экземпляра данной переменной (того же типа и размера) для каждой нити. Изменение нитью значения своей локальной переменной никак не влияет на изменение значения этой же локальной переменной в других нитях.

Если несколько переменных одновременно записывают значение общей переменной без выполнения синхронизации или если как минимум одна нить читает значение общей переменной и как минимум одна нить записывает значение этой переменной без выполнения синхронизации, то возникает ситуация «гонки данных» (data race), при которой результат выполнения программы непредсказуем.

Явно назначить класс переменных по умолчанию можно с помощью опции default.

default (shared | private | none)

Не рекомендуется постоянно полагаться на правила по умолчанию, для большей надёжности лучше всегда явно описывать классы используемых переменных, указывая в директивах OpenMP опции private, shared, firstprivate, lastprivate, reduction.

shared

```
int main(int argc, char *argv[])
{
   int iShared=0;
   #pragma omp parallel num_threads(8)
      {
         iShared = omp_get_thread_num();
         cout << iShared << endl;
      }
   cout << " Последний поток " << iShared <<
endl;
}
```

```
42

50
6

1
7
3
 Последний поток 7
```

private

```
int main(int argc, char *argv[])
{
   #pragma omp parallel num_threads(8)
      {
      int iPrivate = omp_get_thread_num();
      cout << iPrivate << endl;
      }
   cout << " Последний поток ?"  << endl;
}
```

//iPrivate was not declared here

Пример shared

```
int main()
{
   int i, m[10];
  cout << "Массив m в начале: ";
   /* Заполним массив m нулями и напечатаем его */
   for (i = 0; i < 10; i++){
      m[i]=0;
     cout << 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;
   }
   cout << endl << "Массив m в конце: " ;
   for (i = 0; i < 10; i++) cout <<  m[i] << " ";
}
```

Массив m – общий для всех нитей.
В параллельной области каждая нить находит элемент, номер которого совпадает с порядковым номером нити в общем массиве, и присваивает этому элементу значение 1.
В последовательной области печатается изменённый массив.

```
Массив m в начале: 0 0 0 0 0 0 0 0 0 0
Массив m в конце: 1 1 1 1 0 0 0 0 0 0
```

Статические (static) переменные, определённые в параллельной области программы, являются общими (shared). Динамически выделенная память также является общей, однако указатель на неё может быть как общим, так и локальным.

Опции private и firstprivate

- 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;

```
int main()
{
  int iCount=5;
    #pragma omp parallel private(iCount) num_threads(4)
    {
        cout << "In parallel section: " + to_string(iCount) + " \n";
    }
    cout << " After parallel section: " << iCount << endl;
}
```

firstprivate(iCount)

```
In parallel section: 5
In parallel section: 5
In parallel section: 5
In parallel section: 5
 After parallel section: 5
```

```
In parallel section: 0
In parallel section: 0
In parallel section: 0
In parallel section: 32675
 After parallel section: 5
```

Пример firstprivate

Отдельные правила определяют назначение классов переменных при входе и выходе из параллельной области или параллельного цикла при использовании опций reduction, firstprivate, lastprivate, copyin.

```
int main()
{
   int n=1;
   cout << "Значение n в начале: " << n << endl;
#pragma omp parallel firstprivate(n)
   {
      cout << "Значение n на нити (на входе): " << n << endl;
      /* Присвоим переменной n номер текущей нити */
      n = omp_get_thread_num();
      cout << "Значение n на нити (на выходе): " << n << endl;
   }
   cout << "Значение n в конце: " << n;
}
```

```
Значение n в начале: 1
Значение n на нити (на входе): 1
Значение n на нити (на выходе): 0
Значение n на нити (на входе): 1
Значение n на нити (на выходе): 1
Значение n на нити (на входе): 1
Значение n на нити (на выходе): 3
Значение n на нити (на входе): 1
Значение n на нити (на выходе): 2
Значение n в конце: 1
```

В параллельной области все нити :

- вывели значение своей копии переменной n в начале параллельной области равной 1;
- присвоили переменной n свой порядковый номер и ещё раз вывели значение n.

Директива threadprivate

threadprivate указывает, что переменные из списка должны быть размножены с тем, чтобы каждая нить имела свою локальную копию.

#pragma omp threadprivate(список)

Директива threadprivate может позволить сделать локальные копии для статических переменных, которые по умолчанию являются общими.

Для корректного использования локальных копий глобальных объектов нужно гарантировать, что они используются в разных частях программы одними и теми же нитями.

Переменные, объявленные как threadprivate, не могут использоваться в опциях директив OpenMP, кроме copyin, copyprivate, schedule, num_threads, if.

Пример threadprivate

```
int n;
#pragma omp threadprivate(n)
   int main() {
   n = 1;
   cout << " In the start: " << n << endl;
   omp_set_num_threads(4);
   #pragma omp parallel
      {
         n = omp_get_thread_num();
         cout << "First parallel section: thread №" << omp_get_thread_num() << " n=" << n << endl;
      }
   cout << " In the middle: " << n << endl;
   #pragma omp parallel
   {
      cout << "Second parallel section: thread №" << omp_get_thread_num() << " n=" << n << endl;
   }
}
```

```
 In the start: 1
First parallel section: thread №0 n=0
First parallel section: thread №2 n=2
First parallel section: thread №1 n=1
First parallel section: thread №3 n=3
 In the middle: 0
Second parallel section: thread №2 n=2
Second parallel section: thread №1 n=1
Second parallel section: thread №0 n=0
Second parallel section: thread №3 n=3
```

Пример copyin

Переменная, объявленная как threadprivate, инициализирована значением размножаемой переменной из нити-мастера, на входе в параллельную область использована опция copyin.

Глобальная переменная n определена как threadprivate.

Применение copyin позволяет инициализировать локальные копии переменной n начальным значением нити-мастера.

Все нити выведут значение n, равное 1.

```
1  #include <iostream>
2  #include <omp.h>
3  using namespace std;
4  int n;
5  #pragma omp threadprivate(n)
6  int main(int argc, char *argv[])
7  {
8     n = 1;
9  #pragma omp parallel copyin(n)
10    {
11       cout << "Значение n: " << n << endl;
12    }
13 }
```

Распределение работы

OpenMP предлагает несколько вариантов распределения работы между запущенными нитями. Конструкции распределения работ в OpenMP не порождают новых нитей.

Низкоуровневое распараллеливание

Все нити в параллельной области нумеруются последовательными целыми числами от 0 до N – 1 (N – количество нитей в данной области). Можно программировать на самом низком уровне, распределяя работу с помощью функций omp_get_thread_num() и omp_get_num_threads().

Вызов функции omp_get_thread_num() позволяет нити получить свой уникальный номер в текущей параллельной области **int omp_get_thread_num(void);**

Вызов функции omp_get_num_threads() позволяет нити получить количество нитей в текущей параллельной области **int omp_get_num_threads(void);**

В этом случае программист должен явно организовывать синхронизацию доступа к общим данным.

Другие способы распределения работ в OpenMP обеспечивают значительную часть этой работы автоматически.

Параллельные циклы

Если в параллельной области встретился оператор цикла, то, согласно общему правилу, он будет выполнен всеми нитями текущей группы, то есть каждая нить выполнит все итерации данного цикла.

Для распределения итераций цикла между различными нитями можно использовать директиву for.

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

Эта директива относится к идущему следом за данной директивой блоку, включающему операторы for.

Возможные опции:

private(список);

firstprivate(список);

lastprivate(список);

reduction(оператор:список);

Возможные опции:

- schedule(type[, chunk]) – опция задаёт, каким образом итерации цикла распределяются между нитями;
- collapse(n) – опция указывает, что n последовательных тесновложенных циклов ассоциируется с данной директивой; для циклов образуется общее пространство итераций, которое делится между нитями; если опция collapse не задана, то директива относится только к одному непосредственно следующему за ней циклу;
- ordered – сообщает, что в цикле могут встречаться директивы ordered; в этом случае определяется блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации идут в последовательном цикле;
- nowait – убирает неявную барьерную синхронизацию, которая по умолчанию стоит в конце цикла.

Параллельные циклы

На вид параллельных циклов накладываются достаточно жёсткие ограничения.

Предполагается, что корректная программа не должна зависеть от того, какая именно нить какую итерацию параллельного цикла выполнит.

Нельзя использовать побочный выход из параллельного цикла.

Размер блока итераций, указанный в опции schedule, не должен изменяться в рамках цикла.

Формат параллельных циклов:

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

OpenMP должен при входе в цикл точно определить число итераций.

Если директива параллельного выполнения стоит перед гнездом циклов, завершающихся одним оператором, то директива действует только на самый внешний цикл.

Итеративная переменная распределяемого цикла должна быть локальной, если она специфицирована общей, то она неявно делается локальной при входе в цикл. После завершения цикла значение итеративной переменной цикла не определено, если она не указана в опции lastprivate.

Пример for

```
int main(int argc, char *argv[])
{
   int A[10], B[10], C[10], i, n;
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)
   {
            n = omp_get_thread_num(); // номер текущей нити
#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);
      }
   }
}
```

В параллельной области массивы объявлены общими. Вспомогательные переменные – локальными.
n – порядковый номер нити.
Итерации цикла распределены между существующими нитями. Указан номер нити, выполнившей данную итерацию.

```
Нить 1 сложила элементы с номером 3
Нить 1 сложила элементы с номером 4
Нить 1 сложила элементы с номером 5
Нить 2 сложила элементы с номером 6
Нить 2 сложила элементы с номером 7
Нить 0 сложила элементы с номером 0
Нить 0 сложила элементы с номером 1
Нить 0 сложила элементы с номером 2
Нить 3 сложила элементы с номером 8
Нить 3 сложила элементы с номером 9
```

Вычисление суммы элементов массива

Последовательное

```
int sum_arr(int *a, int n) {
  int sum = 0;
  for (int i = 0; i < n; ++i)
    sum += a[i];
  return sum;
}
```

Если каждый поток вычисляет часть суммы, потоки должны синхронизироваться и объединить результаты (модель threads).

Параллельное

```
int sum_arr(int *a, const int n) {
  int sum = 0;
#pragma omp parallel shared(a) reduction (+: sum) num_threads(2)
  {
# pragma omp for
    for (int i = 0; i < n; ++i)
      sum += a[i];
  }
  return sum;
}
```

Использовано 2 потока.
a – общая переменная, i, sum – локальные.

Преимущество OpenMP – сначала написать и отладить последовательную программу (отлаживать параллельную программу тяжело), затем постепенно распараллеливать, дополняя директивами OpenMP.

Маленькие хитрости

Вопрос: Чем отличаются результаты выполнения циклов, коды которых приведены ниже?

Цикл 1 (с прагмой parallel for):

```
1  int i;
2  #pragma omp parallel for
3  for (i='a'; i<='z'; i++)
4  printf ("%c", i);
```

https://software.intel.com/ru-ru/articles/more-work-sharing-with-openmp/

Цикл 2 (с прагмой parallel):

```
1  int i;
2  #pragma omp parallel
3  for (i='a'; i<='z'; i++)
4  printf ("%c", i);
```

Ответ: При выполнении первого цикла на экран однократно будет выведен английский алфавит, а при выполнении второго цикла алфавит будет выведен неопределенное число раз (или столько раз, сколько организовано потоков, если переменная i была объявлена индивидуальной.

Параллельные секции

Директива sections (sections ... end sections) используется для задания конечного (неитеративного) параллелизма.

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

Директива определяет набор независимых секций кода, каждая из которых выполняется своей нитью.

Возможные опции:

private(список); firstprivate(список); lastprivate(список); reduction(оператор:список); nowait.

Директива section задаёт участок кода внутри секции sections для выполнения одной нитью.

#pragma omp section

Перед первым участком кода в блоке sections директива section не обязательна. Какие именно нити будут задействованы для выполнения какой секции, не специфицируется. Если количество нитей больше количества секций, то часть нитей для выполнения данного блока секций не будет задействована. Если количество нитей меньше количества секций, то некоторым (или всем) нитям достанется более одной секции.

Пример sections

```
 4  int main(int argc, char *argv[])
 5  {
 6     int n;
 7  #pragma omp parallel private(n)
 8     {
 9        n=omp_get_thread_num();
10  #pragma omp sections
11        {
12  #pragma omp section
13           {
14              printf("Первая секция, процесс %d\n", n);
15           }
16  #pragma omp section
17           {
18              printf("Вторая секция, процесс %d\n", n);
19           }
20  #pragma omp section
21           {
22              printf("Третья секция, процесс %d\n", n);
23           }
24        }
25        printf("Параллельная область, процесс %d\n", n);
26     }
27  }
```

Три нити, на которые распределились три секции section, выведут сообщение со своим номером, потом все нити напечатают одинаковое сообщение со своим номером.

```
Первая секция, процесс 0
Третья секция, процесс 3
Вторая секция, процесс 2
Параллельная область, процесс 3
Параллельная область, процесс 0
Параллельная область, процесс 2
Параллельная область, процесс 1
```

Пример lastprivate

```
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;
         }
      }
      cout << "Значение n на нити " << omp_get_thread_num() <<  n << endl;
   }
   cout << "Значение n в последовательной области: " << n;
}
```

Опция lastprivate используется вместе с директивой sections.

Переменная n объявлена как lastprivate.

Три нити, выполняющие секции section, присваивают своей локальной копии n разные значения.

По выходе из sections значение n из последней секции присваивается локальным копиям во всех нитях, поэтому все нити напечатают число 3. Это же значение сохранится для переменной n и в последовательной области.

```
Значение n на нити 33
Значение n на нити 23
Значение n на нити 03
Значение n на нити 13
Значение n в последовательной области: 3
```

Задачи (tasks)

Директива task применяется для выделения отдельной независимой задачи.

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

Текущая нить выделяет в качестве задачи ассоциированный с директивой блок операторов.

Задача может выполняться немедленно после создания или быть отложенной на неопределённое время и выполняться по частям. Размер таких частей, а также порядок выполнения частей разных отложенных задач определяется реализацией.

Возможные опции: if(условие);

untied – в случае откладывания задача может быть продолжена любой нитью из числа выполняющих данную параллельную область; если опция не указана, то задача может быть продолжена только породившей её нитью;

default(private|firstprivate|shared|none) – всем переменным в задаче, которым явно не назначен класс, будет назначен класс private, firstprivate или shared;

none – всем переменным в задаче класс должен быть назначен явно, варианты shared или none;

private(список); firstprivate(список); shared(список)

Для гарантированного завершения в точке вызова всех запущенных задач используется директива taskwait.

#pragma omp taskwait

Пример task

```
#define NTASKS 5
void work(int num){
  printf("Процесс %d ожидает\n", num);
  sleep(1);
}
int main(int argc, char *argv[])
{
  omp_lock_t lock;
  int n=NTASKS, i, num;
  omp_init_lock(&lock);
#pragma omp parallel private(i, num)
  {
    num=omp_get_thread_num();
    for(i = 0; i < n; i++)
#pragma omp task
    {
      work(num);
      while(!omp_test_lock(&lock)){
#pragma omp taskyield
      }
      printf("Процесс %d в критической секции\n", num);
      //sleep(1);
      omp_unset_lock(&lock);
    }
  }
```

Нить, выполнившая данную директиву, приостанавливается до тех пор, пока не будут завершены все ранее запущенные данной нитью независимые задачи.

```
Процесс 0 ожидает
Процесс 0 ожидает
Процесс 0 ожидает
Процесс 0 ожидает
Процесс 0 в критической секции
Процесс 0 в критической секции
Процесс 1 ожидает
Процесс 0 в критической секции
Процесс 1 ожидает
Процесс 0 ожидает
Процесс 0 в критической секции
Процесс 1 ожидает
Процесс 1 в критической секции
Процесс 1 ожидает
Процесс 0 в критической секции
Процесс 1 в критической секции
Процесс 2 ожидает
Процесс 1 ожидает
Процесс 1 в критической секции
Процесс 2 ожидает
```

Вопросы

- Может ли программа на OpenMP состоять только из параллельных областей? Только из последовательных областей?
- Чем отличается нить-мастер от всех остальных нитей?
- В каких случаях может быть необходимо использование опции if директивы parallel?
- При помощи трёх уровней вложенных параллельных областей породите 8 нитей (на каждом уровне параллельную область должны исполнять 2 нити). Посмотрите, как будет исполняться программа, если запретить вложенные параллельные области.
- Чем отличаются директивы single и master? Может ли нить-мастер выполнить область, ассоциированную с директивой single?
- Может ли нить с номером 1 выполнить область, ассоциированную с директивой master?
- Может ли одна и та же переменная выступать в одной части программы как общая, а в другой части – как локальная? Что произойдёт, если несколько нитей одновременно обратятся к общей переменной?
- Может ли произойти конфликт, если несколько нитей одновременно обратятся к одной и той же локальной переменной?
- Каким образом при входе в параллельную область разослать всем порождаемым нитям значение некоторой переменной?
- Можно ли сохранить значения локальных копий общих переменных после завершения параллельной области? Если да, то что необходимо для их использования?
- В чём отличие опции copyin от опции firstprivate?
- Могут ли функции omp_get_thread_num() и omp_get_num_threads() вернуть одинаковые значения на нескольких нитях одной параллельной области?
- Можно ли распределить между нитями итерации цикла без использования директивы for?
- Можно ли одной директивой распределить между нитями итерации сразу нескольких циклов?
- Возможно ли, что при статическом распределении итераций цикла нитям достанется разное количество итераций?
- Могут ли при повторном запуске программы итерации распределяемого цикла достаться другим нитям? Если да, то при каких способах распределения итераций?
- Можно ли реализовать параллельные секции без использования директив sections и section?
- Как при выходе из параллельных секций разослать значение некоторой локальной переменной всем нитям, выполняющим данную параллельную область?
- В каких случаях может пригодиться механизм задач?