Лабораторная работа №3
Модель данных

Модель данных в OpenMP предполагает наличие как общей для всех нитей области памяти, так и локальной области памяти для каждой нити.
В OpenMP переменные в параллельных областях программы разделяются на два основных класса:
· `shared` (общие; все нити видят одну и ту же переменную);
· `private` (локальные, приватные; каждая нить видит свой экземпляр данной переменной).
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 так называемой «гонки данных» (data race), при которой результат выполнения программы непредсказуем.
По умолчанию, все переменные, порождённые вне параллельной области, при входе в эту область остаются общими (`shared`). Исключение составляют переменные, являющиеся счетчиками итераций в цикле, по очевидным причинам. Переменные, порождённые внутри параллельной области, по умолчанию являются локальными (`private`). Явно назначить класс переменных по умолчанию можно с помощью опции `default`. 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`.
Пример 12 демонстрирует использование опции private. В данном примере переменная n объявлена как локальная переменная в параллельной области. Это значит, что каждая нить будет работать со своей копией переменной n, при этом в начале параллельной области на каждой нити переменная n не будет инициализирована. В ходе выполнения программы значение переменной n будет выведено в четырёх разных местах. Первый раз значение n будет выведено в последовательной области, сразу после присваивания переменной n

30 значения 1. Второй раз все нити выведут значение своей копии переменной n в начале параллельной области, неинициализированное значение может зависеть от реализации. Далее все нити выведут свой порядковый номер, полученный с помощью функции `omp_get_thread_num()` и присвоенный переменной n. После завершения параллельной области будет ещё раз выведено значение переменной n, которое окажется равным 1 (не изменилось во время выполнения параллельной области).

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
int n=1;
printf("n в последовательной области (начало): %d\n",
n);
#pragma omp parallel private(n)
{
printf("Значение n на нити (на входе): %d\n", n);
/* Присвоим переменной n номер текущей нити */
n=omp_get_thread_num();
printf("Значение n на нити (на выходе): %d\n", n);
}
printf("n в последовательной области (конец): %d\n",
n);
}
```

Пример 12. Опция private на языке Си.

Пример 13 демонстрирует использование опции `shared`. Массив m объявлен общим для всех нитей. В начале последовательной области массив m заполняется нулями и выводится на печать. В параллельной области каждая нить находит элемент, номер которого совпадает с порядковым номером нити в общем массиве, и присваивает этому элементу значение 1. Далее, в последовательной области печатается изменённый массив m.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
int i, m[10];
printf("Массив m в начале:\n");
/* Заполним массив m нулями и напечатаем его */
```

```
for (i=0; i<10; i++){
m[i]=0;
printf("%d\n", 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;
}
/* Ещё раз напечатаем массив */
printf("Массив m в конце:\n");
for (i=0; i<10; i++) printf("%d\n", m[i]);
}
```
Пример 13. Опция shared на языке Си.

В языке Си 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.
Отдельные правила определяют назначение классов переменных при входе и выходе из параллельной области или параллельного цикла при использовании опций `reduction`, `firstprivate`, `lastprivate`, `copyin`.
Пример 14 демонстрирует использование опции `firstprivate`. Переменная n объявлена как `firstprivate` в параллельной области. Значение n будет выведено в четырёх разных местах. Первый раз значение n будет выведено в последовательной области сразу после инициализации. Второй раз все нити выведут значение своей копии переменной n в начале параллельной области, и это значение будет равно 1. Далее, с помощью функции `omp_get_thread_num()` все нити присвоят переменной n свой порядковый номер и ещё раз выведут значение n. В последовательной области будет ещё раз выведено значение n, которое снова окажется равным 1.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
int n=1;
printf("Значение n в начале: %d\n", n);
#pragma omp parallel firstprivate(n)
{
```

```
printf("Значение n на нити (на входе): %d\n", n);
/* Присвоим переменной n номер текущей нити */
n=omp_get_thread_num();
printf("Значение n на нити (на выходе): %d\n", n);
}
printf("Значение n в конце: %d\n", n);
}
```

Пример 14. Опция `firstprivate` на языке Си.

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

```
#pragma omp threadprivate(список)
```

Директива `threadprivate` может позволить сделать локальные копии для статических переменных языка Си и COMMON-блоков языка Фортран, 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. Если на локальные копии ссылаются в разных параллельных областях, то для сохранения их значений необходимо, чтобы не было объемлющих параллельных областей, количество нитей в обеих областях совпадало, а переменная `OMP_DYNAMIC` была установлена в `false` с начала первой области до начала второй. Переменные, объявленные как `threadprivate`, не могут использоваться в опциях директив OpenMP, кроме `copyin`, `copyprivate`, `schedule`, `num_threads`, `if`.

Пример 15 демонстрирует использование директивы `threadprivate`. Глобальная переменная n объявлена как `threadprivate` переменная. Значение переменной n выводится в четырёх разных местах. Первый раз все нити выведут значение своей копии переменной n в начале параллельной области, и это значение будет равно 1 на нити-мастере и 0 на остальных нитях. Далее с помощью функции `omp_get_thread_num()` все нити присвоят переменной n свой порядковый номер и выведут это значение. Затем в последовательной области будет ещё раз выведено значение переменной n, которое окажется равным порядковому номеру нити-мастера, то есть 0. В последний раз значение переменной n выводится в новой параллельной области, причём значение каждой локальной копии должно сохраниться.

```
#include <stdio.h>
#include <omp.h>
int n;
#pragma omp threadprivate(n)
```

```
int main(int argc, char *argv[])
{
int num;
n=1;
#pragma omp parallel private (num)
{
num=omp_get_thread_num();
printf("Значение n на нити %d (на входе): %d\n", num,
n);
/* Присвоим переменной n номер текущей нити */
n=omp_get_thread_num();
printf("Значение n на нити %d (на выходе): %d\n", num,
n);
}
printf("Значение n (середина): %d\n", n);
#pragma omp parallel private (num)
{
num=omp_get_thread_num();
printf("Значение n на нити %d (ещё раз): %d\n", num,
n);
}
}
```

Пример 15. Директива `threadprivate` на языке Си.

Если необходимо переменную, объявленную как `threadprivate`, инициализировать значением размножаемой переменной из нити-мастера, то на входе в параллельную область можно использовать опцию `copyin`. Если значение локальной переменной или переменной, объявленной как `threadprivate`, необходимо переслать от одной нити всем, работающим в данной параллельной области, для этого можно использовать опцию `copyprivate` директивы `single`.

Пример 16 демонстрирует использование опции `copyin`. Глобальная переменная n определена как `threadprivate`. Применение опции `copyin` позволяет инициализировать локальные копии переменной n начальным значением нити-мастера. Все нити выведут значение n, равное 1.

```
#include <stdio.h>
int n;
#pragma omp threadprivate(n)
int main(int argc, char *argv[])
{
```

```
n=1;
#pragma omp parallel copyin(n)
{
printf("Значение n: %d\n", n);
}
}
```

Пример 16. Опция `copyin` на языке Си.

Задания

1. Может ли одна и та же переменная выступать в одной части программы как общая, а в другой части – как локальная?
2. Что произойдёт, если несколько нитей одновременно обратятся к общей переменной?
3. Может ли произойти конфликт, если несколько нитей одновременно обратятся к одной и той же локальной переменной?
4. Каким образом при входе в параллельную область разослать всем порождаемым нитям значение некоторой переменной?
5. Можно ли сохранить значения локальных копий общих переменных после завершения параллельной области? Если да, то что необходимо для их использования?
6. В чём отличие опции `copyin` от опции `firstprivate`?