Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное

образовательное учреждение высшего образования

«Южно-Уральский государственный университет

(национальный исследовательский университет)»

Высшая школы электроники и компьютерных наук

Кафедра системного программирования

ОТЧЕТ  
о лабораторной работе №2  
по дисциплине «Технологии параллельного программирования»

Выполнил:   
студент группы КЭ-220   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Голенищев А. Б.   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.   
   
Отчет принял:   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Жулев А. Э.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

***Задание 4. Общие и частные переменные в OpenMP: программа   
«Скрытая ошибка»***

Представлен вариант кода, где rank является частной переменной, листнинг 1.

#include <stdio.h>

#include <omp.h>

// Golenishchev Artem, KE-220 Task 4

int main() {

int k;

printf("Enter number of threads (k): ");

scanf("%d", &k);

omp\_set\_num\_threads(k);

printf("Rank is shared: \n");

// Параллельная область с общей переменной rank

#pragma omp parallel shared(k)

{

int rank = omp\_get\_thread\_num();

printf("I am %d thread.\n", rank);

}

printf("\nRank is private: \n");

// Параллельная область с частной переменной rank

#pragma omp parallel

{

int rank = omp\_get\_thread\_num(); // Локальная переменная rank

printf("I am %d thread.\n", rank);

}

return 0;

}

Листнинг 1. Код, где rank имеет разные области видимости

Результат выполнения программы представлен на рисунке 1.

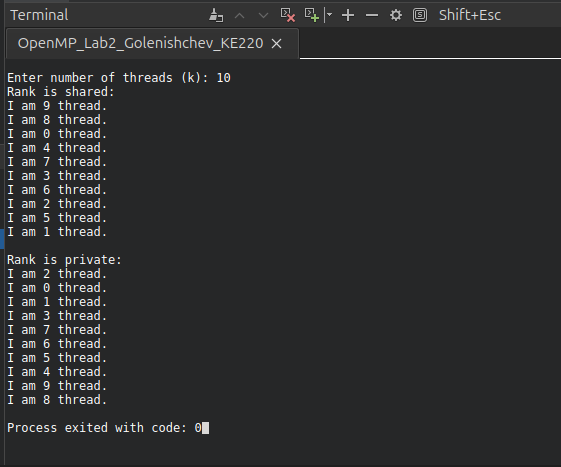


Рисунок 1. Результат выполнения программы в Qt Creator

На рисунке 1 видно, что в случае области видимости переменной private(rank) потоки с выводом через printf() одновременно работают, из-за чего мы наблюдаем некорректный вывод. Скрытая ошибка должна заключаться в повторении несколькими потоками одного и того же значения в выводе rank в случае области видимости переменной shared(rank). Для поиска скрытой ошибки повторим запуск еще несколько раз, увеличивая значение omp\_set\_num\_threads до количества потоков нашего процессора (24) или больше его. Кроме того, запускать код без изменений бесполезно, для фиксации момента повторений можно использовать sleep() или синхронизацию. Модифицируем код, листнинг 2 Запустим вариант кода для поиска скрытой ошибки, рисунки 2 Суть скрытой ошибки заключается в том, что при многопоточности значение переменной rank успевает перезаписываться до вывода на экран (гонка данных). Cпособ с использованием sleep() позволяет зафиксировать гонку данных (ее наличие).

Листнинг 2. Код, где мы ловим ошибку с помощью sleep()

#include <stdio.h>

#include <omp.h>

#include <unistd.h> // Для использования usleep() в Ubuntu

// Golenishchev Artem, KE-220 Task 4

int main() {

int k;

// Ввод количества нитей

printf("Enter number of threads (k): ");

scanf("%d", &k);

// Установка количества нитей

omp\_set\_num\_threads(k);

int rank;

// Параллельная область с указанием области видимости

printf("Rank is shared: \n");

#pragma omp parallel shared(rank)

{

rank = omp\_get\_thread\_num(); // Переменная rank общая для всех потоков

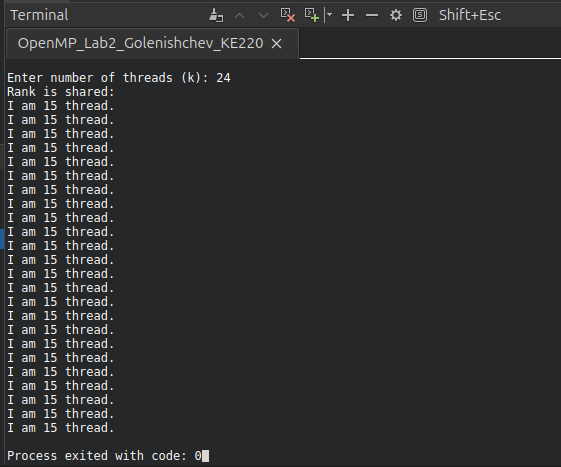
usleep(100000); // Имитация длительных вычислений (100000 микросекунд = 100 мс)

printf("I am %d thread.\n", rank);

}

return 0;

}

Рисунок 2. Результат выполнения программы с sleep()

***Задание 5. Общие и частные переменные в OpenMP: параметр reduction***

Код программы представлен ниже, листнинг 3. Показан результат работы программы, рисунок 3.

#include <stdio.h>

#include <omp.h>

// Golenishchev Artem, KE-220 Task 5

int main() {

int n, k;

// Ввод числа N и количества нитей k

printf("Enter an integer N: ");

scanf("%d", &n);

printf("Enter an integer number of threads k: ");

scanf("%d", &k);

int sum = 0; // Общая сумма

// Установка количества нитей

omp\_set\_num\_threads(k);

printf("Partial sums:\n");

#pragma omp parallel reduction(+:sum)

{

int thread\_id = omp\_get\_thread\_num(); // Номер текущей нити

int total\_threads = omp\_get\_num\_threads(); // Общее количество нитей

// Диапазон чисел для текущей нити

int start = (n \* thread\_id) / total\_threads + 1;

int end = (n \* (thread\_id + 1)) / total\_threads;

int local\_sum = 0; // Частичная сумма для текущей нити

for (int i = start; i <= end; ++i) {

local\_sum += i;

}

// Локальная сумма автоматически добавляется в sum через reduction

sum += local\_sum;

// Вывод частичной суммы для текущей нити

printf("[%d]: Sum = %d\n", thread\_id, local\_sum);

}

// Вывод общей суммы

printf("Result: Sum = %d\n", sum);

return 0;

}

Листнинг 3. Программа вычисления суммы чисел, работающая на k-нитей

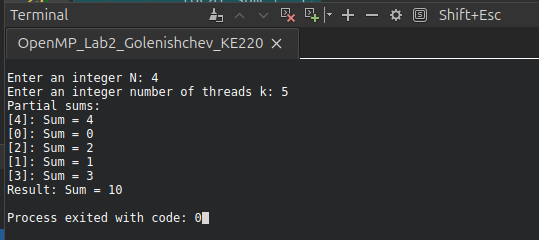


Рисунок 3. Результат выполнения программы расчета суммы чисел

Программа демонстрирует использование OpenMP для выполнения параллельных вычислений с заданным числом потоков, введённым пользователем. В каждом потоке определяется его уникальный идентификатор (rank), который затем используется для вывода сообщения. Функция usleep добавляет искусственную задержку, имитируя длительные вычисления, чтобы подчеркнуть асинхронность работы потоков.

***Ответы на вопросы к лабораторной работе:***

1. Для чего нужны частные переменные? Не противоречит ли их существование реализуемой OpenMP модели программирования в общей памяти? Приведите содержательный пример частной переменной.

Частные переменные необходимы, чтобы каждому потоку выделять свою локальную копию данных, что предотвращает конфликты и обеспечивает независимость вычислений. Их существование не противоречит модели общей памяти OpenMP, так как локальные копии переменных размещаются в памяти отдельных потоков.

#pragma omp parallel for private(i)

for (i = 0; i < N; ++i) {

    // Каждая нить использует локальную копию переменной i

}

1. Какие новые области видимости появляются в параллельной программе? Как они задаются?

В параллельном программировании shared() хранит данные в сегменте стека главной нити, а private() внутри сегмента стека нити.



1. Продемонстрируйте конфликт обращений к переменной rank в написанной программе? Всегда ли он возникает? Как его предотвратить?

Конфликт возникает, если несколько потоков одновременно читают и записывают одно и то же значение переменной rank. Это приводит к неопределённому поведению. Конфликт не возникнет, если потоки не пересекаются по времени выполнения или используется синхронизация. Предотвращается private(rank).

int rank = 0;

#pragma omp parallel

{

    rank += omp\_get\_thread\_num(); // Одновременная запись несколькими потоками

    printf("Thread %d: rank = %d\n", omp\_get\_thread\_num(), rank);

}

1. Как оформляются в программе входные данные, промежуточные переменные и результаты выполнения параллельного региона?

Входные данные: Передаются в параллельную область как shared, чтобы они были доступны всем потокам.

Промежуточные переменные: Обычно объявляются как private, чтобы потоки выполняли независимые вычисления.

Результаты: Объединяются через reduction или синхронизируются с помощью директивы critical.

Пример:

#pragma omp parallel reduction(+:sum)

{

    int local\_sum = 0;

    for (int i = start; i < end; i++) {

        local\_sum += i;

    }

    sum += local\_sum; // Частичная сумма добавляется к общей.

}

1. Какие дополнительные действия выполняет директива, если она имеет параметр reduction? Чем инициализируются частные переменные, создаваемые параметром reduction?

Директива reduction автоматически:

* 1. Создаёт частные копии переменной для каждого потока.
  2. Инициализирует эти переменные в зависимости от операции   
     (например, 0 для +, 1 для \*).
  3. Выполняет объединение частных значений в конце параллельного региона.

Пример:

int sum = 0;

#pragma omp parallel for reduction(+:sum)

for (int i = 1; i <= 10; i++) {

    sum += i; // Каждый поток работает с локальной копией sum.

}

// В конце значения объединяются, и sum содержит итоговую сумму.

***Выводы:***

Изучили основные механизмы управления потоками в OpenMP, такие как использование общих (shared) и частных (private) переменных, распределение задач между потоками, а также предотвращение конфликтов при одновременном доступе к данным. Особое внимание было уделено параметру reduction, который позволяет автоматически создавать локальные копии переменных, выполнять параллельные вычисления и корректно объединять результаты. На практике было продемонстрировано, как эффективно распределять вычисления между потоками, минимизировать ошибки доступа к памяти и организовать синхронный сбор результатов для получения корректного результата.