МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по лабораторной №5

по дисциплине «Системное программирование»

**Технология OpenMP**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: | студент гр. ФИБ-3301-51-01 |  | / В. Р. Кочкин / |
| Проверил: | к.ф.-м.н. доцент каф. ПМиИ |  | / В. А. Бызов / |

Киров 2022

Цель работы

Получить навыки разработки параллельных алгоритмов с использованием технологии OpenMP.

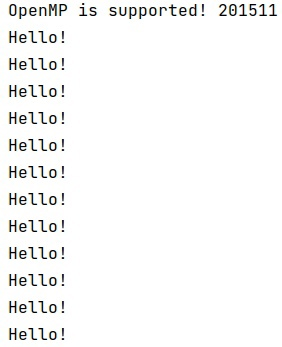
Задания

**Задание 0**

Проверить, поддерживается ли OpenMP. Если нет, то включить поддержку в

параметрах проекта.

**Полученные результаты**

Листинг программы приведен в [приложении А задание 0](#_Задание_0.).

**Задание 1**

Скопировать фрагмент кода в программу. Запустить на выполнение.

#include <omp.h>

int main () {

int nthreads, tid;

// Создание параллельной области

#pragma omp parallel private(tid)

{

// печать номера потока

tid = omp\_get\_thread\_num();

printf("Hello World from thread = %d\n", tid);

// Печать количества потоков – только master

if (tid == 0) {

nthreads = omp\_get\_num\_threads();

printf("Number of threads = %d\n", nthreads);

}

15

} // Завершение параллельной области

}

Ответить на вопросы:

а) Зачем нужна директива parallel?

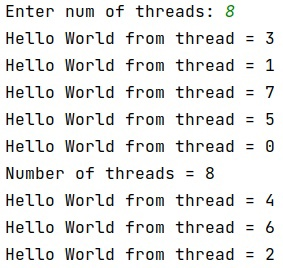
б) Сколько потоков было запущено? Почему?

в) Сколько потоков одновременно работают с переменной tid? Почему?

г) Поток с каким tid останется после завершения параллельной области?

Запросить требуемое количество потоков у пользователя. Задать количество потоков для параллельной области.

**Полученные результаты**

 **Ответы на вопросы**

1. Директива parallel нужна для создания параллельной области программы;
2. Было запущено 12 потоков, так как по умолчанию количество потоков равняется количеству логических ядер процессора;
3. Переменная tid копируется в каждый поток, поэтому все потоки работают с разными копиями переменной;
4. После завершения параллельной области останется поток с tid = 0, так как это главный поток программы main.

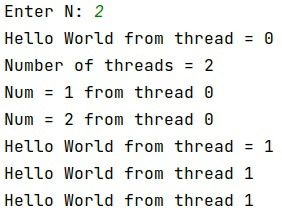
Листинг программы приведен в [приложении А задание 1](#_Задание_1.).

**Задание 2**

Написать программу, задающую работу двух потоков. Первый поток в цикле выводит последовательно числа от 1 до N, а второй – N раз выводит слово

«HELLO». Число N задаётся пользователем.

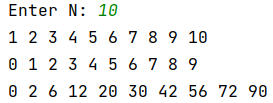
**Полученные результаты**

Листинг программы приведен в [приложении А задание 2](#_Задание_2.).

**Задание 3**

Написать параллельную программу, находящую поэлементное произведение двух массивов размера N. Задать параметр schedule, попробовать разные аргументы.

**Полученные результаты**

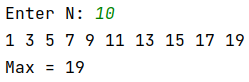
Листинг программы приведен в [приложении А задание 3](#_Задание_3.).

**Задание 4**

Написать параллельную программу, вычисляющую максимальное значение

среди элементов вектора, используя директивы critical.

**Полученные результаты**

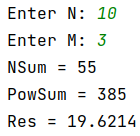
Листинг программы приведен в [приложении А задание 4](#_Задание_4.).

**Задание 5**

Написать программу, задающую работу 𝑀 + 𝐾 потоков. Первые 𝑀 потоков вычисляют сумму от 1 до 𝑁, а оставшиеся 𝐾 потоков вычисляют длину 𝑁-мерного вектора. Число 𝑁 задаётся пользователем.

Указание. Использовать секции и вложенный параллелизм.

**Полученные результаты**

Листинг программы приведен в [приложении А задание 5](#_Задание_5.).

**Задание 6**

Написать параллельную программу, которая каждый элемент вектора размера N заменяет на его наибольший простой делитель. Число N задается пользователем. Элементы вектора – случайные натуральные числа из диапазона [105, 106].

Замерить среднее время выполнения программы для N = 2\*107, 5\*107 и 108 на 1, 2, 4 и 8 потоках. Вычислить среднее ускорение для 2, 4 и 8 потоков. Построить диаграмму зависимости ускорения от числа потоков для каждого размера вектора (3 графика на одной диаграмме).

**Полученные результаты**

*Проверка работы алгоритма.*

*Таблица 5 – Время вычисление суммы, с*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размерность | Число потоков | | | |
| 1 | 2 | 4 | 8 |
| 107 | 0,0023571 | 0,0014051 | 0,001276 | 0,0011502 |
| 108 | 0,0144072 | 0,0077773 | 0,005149 | 0,004044 |
| 109 | 0,135902 | 0,0690642 | 0,05326 | 0,0318545 |

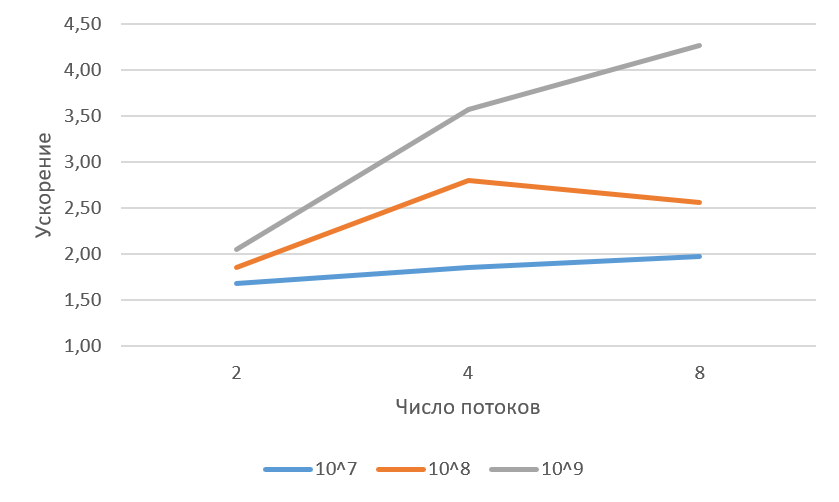
Из таблицы видно, что программа быстрее всего исполнялась на 8 потоках.

*Таблица 6 – Ускорение вычисления суммы, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность | Число потоков | | |
| 2 | 4 | 8 |
| 107 | 1,68 | 1,8472571 | 2,05 |
| 108 | 1,85 | 2,7980035 | 3,56 |
| 109 | 1,97 | 2,551671 | 4,27 |

Из таблицы видно, что максимальное ускорение при 8 потоках, однако ускорение не достигает 8.

*График 3 – Ускорение вычисления суммы, с*



Листинг программы приведен в [приложении А задание 6](#_Задание_6.).

**Задание 7**

Написать параллельную программу, выполняющую умножение двух матриц размера N × N. Разработать программы с использованием распараллеливания циклов разного уровня вложенности.

Замерить среднее время выполнения программ для N = 500, 1000, 2000 на 1, 2, 4 и 8 потоках. Сравнить полученные результаты. Оцените величину накладных расходов на создание и завершение потоков.

Для оптимального варианта вычислить среднее ускорение на 2, 4 и 8 потоках. Построить диаграмму зависимости ускорения от числа потоков для каждого размера матриц (3 графика на одной диаграмме).

**Полученные результаты**

Листинг программы приведен в [приложении А задание 7](#_Задание_7.).

**Задание 8**

Написать параллельную программу, выполняющую поиск максимального значения среди минимальных элементов строк матрицы размера N × N.

Обосновать выбор средств и методов для распараллеливания.

Замерить среднее время выполнения программ для N = 500, 1000, 2000 на 1, 2, 4 и 8 потоках. Вычислить среднее ускорение на 2, 4 и 8 потоках.

**Полученные результаты**

Листинг программы приведен в [приложении А задание 8](#_Задание_8.).

Вывод

В ходе лабораторной работы я получил навыки разработки параллельных алгоритмов с использованием технологии OpenMP. Были произведены замеры времени выполнения программ и на их основании сделаны выводы. Все тесты проводились на процессоре AMD Ryzen 5 3500U 4/8 2.1ГГц.

# **Приложения**

## **Приложение А. Листинги программ**

### Задание 0.

#include <iostream>  
  
int main() {  
 #ifdef **\_OPENMP**  
printf("OpenMP is supported! %d\n", **\_OPENMP**);  
 #else  
 printf("OpenMP is not supported!\n");  
 #endif  
  
 #pragma omp parallel  
 {  
 std::cout << "Hello!\n";  
 }  
  
 return 0;  
}

### Задание 1.

#include <iostream>  
#include <omp.h>  
  
int main() {  
 int nthreads, tid, num\_threads;  
  
 std::cout << "Enter num of threads: ";  
 std::cin >> num\_threads;  
  
 *// Установка количества потоков для программы*  
omp\_set\_num\_threads(num\_threads);  
  
 *// Создание параллельной области*  
#pragma omp parallel private(tid)  
 {  
 *// Печать номера потока*  
tid = omp\_get\_thread\_num();  
 printf("Hello World from thread = %d\n", tid);  
  
 *// Печать количества потоков – только master*  
if (tid == 0) {  
 nthreads = omp\_get\_num\_threads();  
 printf("Number of threads = %d\n", nthreads);  
 }  
 }  
  
 return 0;  
}

### Задание 2.

#include <iostream>  
#include <omp.h>  
  
void thread\_process(int, int);  
  
int main() {  
 int nthreads, tid, N;  
  
 std::cout << "Enter N: ";  
 std::cin >> N;  
  
 *// Установка количества потоков для программы*  
omp\_set\_num\_threads(2);  
  
 *// Создание параллельной области*  
#pragma omp parallel private(tid)  
 {  
 *// Печать номера потока*  
tid = omp\_get\_thread\_num();  
 printf("Hello World from thread = %d\n", tid);  
  
 *// Печать количества потоков – только master*  
if (tid == 0) {  
 nthreads = omp\_get\_num\_threads();  
 printf("Number of threads = %d\n", nthreads);  
 }  
  
 *// Выполнение работы потока*  
thread\_process(tid, N);  
 }  
  
 return 0;  
}  
  
void thread\_process(int tid, int N) {  
 switch(tid) {  
 case 0:  
 for (int i = 0; i < N; ++i) {  
 printf("Num = %d from thread %d\n", i + 1, tid);  
 }  
 break;  
 case 1:  
 for (int i = 0; i < N; ++i) {  
 printf("Hello World from thread %d\n", tid);  
 }  
 break;  
 default:  
 printf("Not support more then 2 threads\n");  
 }  
}

### Задание 3.

#include <iostream>  
#include <omp.h>  
  
int main() {  
 int N;  
 int \*arr\_1, \*arr\_2, \*res;  
  
 std::cout << "Enter N: ";  
 std::cin >> N;  
  
 arr\_1 = new int[N];  
 arr\_2 = new int[N];  
 res = new int[N];  
  
 *//Проинициализируем созданные массивы*  
for (int i = 0; i < N; ++i)  
 {  
 arr\_1[i] = i + 1;  
 arr\_2[i] = i;  
 res[i] = 0;  
 }  
  
 int index;  
  
 *// Создание параллельной области*  
#pragma omp parallel shared(index)  
 {  
 *//Раздадим итерации цикла последовательными порциями*  
#pragma omp for schedule(static)  
 for (index = 0; index < N; ++index)  
 res[index] = arr\_1[index] \* arr\_2[index];  
 }  
  
 for(int i = 0; i < N; i++) {  
 std::cout << arr\_1[i] << " ";  
 }  
 std::cout << std::endl;  
 for(int i = 0; i < N; i++) {  
 std::cout << arr\_2[i] << " ";  
 }  
 std::cout << std::endl;  
 for(int i = 0; i < N; i++) {  
 std::cout << res[i] << " ";  
 }  
 std::cout << std::endl;  
  
 delete[] arr\_1;  
 delete[] arr\_2;  
 delete[] res;  
  
 return 0;  
}

### Задание 4.

#include <iostream>  
#include <omp.h>  
#include <vector>  
  
int main() {  
 int N;  
 std::vector<long> vec;  
  
 std::cout << "Enter N: ";  
 std::cin >> N;  
  
 vec.reserve(N);  
 for (int i = 0; i < N; ++i) {  
 vec.push\_back(2 \* i + 1);  
 std::cout << vec[i] << " ";  
 }  
  
 std::cout << std::endl;  
  
 long max = **INT32\_MIN**;  
 long local\_max = **INT32\_MIN**;  
  
 #pragma omp parallel  
 {  
 #pragma omp for schedule(static)  
 for (int i = 0; i < N; ++i) {  
 if (local\_max < vec[i]) {  
 local\_max = vec[i];  
 }  
 }  
 #pragma omp critical  
 if (max < local\_max) {  
 max = local\_max;  
 }  
 }  
  
 std::cout << "Max = " << max << std::endl;  
  
 return 0;  
}

### Задание 5.

#include <iostream>  
#include <omp.h>  
#include <vector>  
#include <cmath>  
  
int main() {  
 int N, M, K;  
 double res;  
 std::vector<long> vec;  
  
 std::cout << "Enter N: ";  
 std::cin >> N;  
  
 std::cout << "Enter M: ";  
 std::cin >> M;  
  
 K = N - M;  
  
 vec.reserve(N);  
 for (int i = 0; i < N; ++i) {  
 vec.push\_back(i + 1);  
 }  
  
 long long nSum = 0;  
 long localPowSum = 0;  
 long powSum = 0;  
  
 omp\_set\_nested(1);  
 #pragma omp parallel num\_threads(2)  
 {  
 #pragma omp sections  
 {  
 #pragma omp section  
 {  
 #pragma omp parallel num\_threads(M) reduction(+: nSum)  
 {  
 #pragma omp for schedule(static)  
 for (int i = 0; i < N; ++i) {  
 nSum += i + 1;  
 }  
 }  
 }  
 #pragma omp section  
 {  
 #pragma omp parallel num\_threads(K) private(localPowSum)  
 {  
 localPowSum = 0;  
 #pragma omp for schedule(static)  
 for (int i = 0; i < N; ++i) {  
 localPowSum += vec[i] \* vec[i];  
 }  
 #pragma omp critical  
 powSum += localPowSum;  
 #pragma omp flush(powSum)  
 res = sqrt(powSum);  
 }  
 }  
 }  
 }  
  
 std::cout << "NSum = " << nSum << std::endl;  
 std::cout << "PowSum = " << powSum << std::endl;  
 std::cout << "Res = " << res << std::endl;  
  
 return 0;  
}

### Задание 6.

#include <iostream>  
#include <omp.h>  
#include <vector>  
#include <cmath>  
  
long largest\_prime\_divisor(long);  
  
int main() {  
 int N, num\_threads;  
 double start, end, diff;  
 std::vector<long> vec;  
  
 std::cout << "Enter N: ";  
 std::cin >> N;  
  
 std::cout << "Enter num of threads: ";  
 std::cin >> num\_threads;  
  
 vec.reserve(N);  
 for (int i = 0; i < N; ++i) {  
 vec.push\_back(i + 2);  
 std::cout << vec[i] << " ";  
 }  
 std::cout << std::endl;  
  
 start = omp\_get\_wtime();  
  
 #pragma omp parallel num\_threads(num\_threads)  
 {  
 #pragma omp for schedule(static)  
 for (int i = 0; i < N; ++i)  
 vec[i] = largest\_prime\_divisor(vec[i]);  
 }  
  
 end = omp\_get\_wtime();  
 diff = end - start;  
 std::cout << diff << std::endl;  
  
 for (long i : vec) {  
 std::cout << i << " ";  
 }  
 std::cout << std::endl;  
  
 return 0;  
}  
  
long largest\_prime\_divisor(long n) {  
 int maxPrime = -1;  
  
 *// Пока число четное, делим его на 2*  
while (n % 2 == 0) {  
 maxPrime = 2;  
 n /= 2;  
 }  
  
 *// Пока число делится на 3, делим его на 3*  
while (n % 3 == 0) {  
 maxPrime = 3;  
 n /= 3;  
 }  
  
 *// Проходим по натуральным числам, которые не делятся на 2 и 3*  
for (int i = 5; i <= sqrt(n); i += 6) {  
 *// Пока число делится на i, делим его на i*  
while (n % i == 0) {  
 maxPrime = i;  
 n /= i;  
 }  
 *// Пока число делится на i + 2, делим его на i + 2*  
while (n % (i + 2) == 0) {  
 maxPrime = i + 2;  
 n /= i + 2;  
 }  
 }  
  
 *// n - простое число больше 4*  
if (n > 4)  
 maxPrime = n;  
  
 return maxPrime;  
}

### Задание 7.

#include <iostream>  
#include <omp.h>  
#include <vector>  
#include <ctime>  
  
const int RandomLimit = 10;  
const bool Print = true;  
  
std::vector<std::vector<int>> create\_random\_matrix(int);  
std::vector<std::vector<int>> create\_empty\_matrix(int);  
void print\_matrix(const std::vector<std::vector<int>>&);  
  
int main() {  
 srand(time(nullptr));  
  
 int N, num\_threads;  
 double start, end, diff;  
 std::vector<std::vector<int>> matrix\_1, matrix\_2, result;  
  
 std::cout << "Enter N: ";  
 std::cin >> N;  
  
 std::cout << "Enter num of threads: ";  
 std::cin >> num\_threads;  
  
 matrix\_1 = create\_random\_matrix(N);  
 matrix\_2 = create\_random\_matrix(N);  
 result = create\_empty\_matrix(N);  
  
 if (Print) {  
 print\_matrix(matrix\_1);  
 print\_matrix(matrix\_2);  
 }  
  
 start = omp\_get\_wtime();  
  
 #pragma omp parallel num\_threads(num\_threads)  
 {  
 #pragma omp for schedule(static)  
 for (int i = 0; i < N; ++i) {  
 for (int k = 0; k < N; ++k) {  
 for (int j = 0; j < N; ++j) {  
 result[i][j] += matrix\_1[i][k] \* matrix\_2[k][j];  
 }  
 }  
 }  
 }  
  
 end = omp\_get\_wtime();  
 diff = end - start;  
 std::cout << diff << std::endl;  
 std::cout << std::endl;  
  
 if (Print) {  
 print\_matrix(result);  
 }  
  
 return 0;  
}  
  
std::vector<std::vector<int>> create\_random\_matrix(int size) {  
 std::vector<std::vector<int>> matrix;  
  
 for (int i = 0; i < size; ++i) {  
 matrix.emplace\_back();  
 for (int j = 0; j < size; ++j) {  
 matrix[i].push\_back(rand() % RandomLimit);  
 }  
 }  
  
 return matrix;  
}  
  
std::vector<std::vector<int>> create\_empty\_matrix(int size) {  
 std::vector<std::vector<int>> matrix;  
  
 for (int i = 0; i < size; ++i) {  
 matrix.emplace\_back();  
 for (int j = 0; j < size; ++j) {  
 matrix[i].push\_back(0);  
 }  
 }  
  
 return matrix;  
}  
  
void print\_matrix(const std::vector<std::vector<int>>& matrix) {  
 for (const auto& row : matrix) {  
 for (auto column : row) {  
 std::cout << column << " ";  
 }  
 std::cout << std::endl;  
 }  
 std::cout << std::endl;  
}

### Задание 8.

#include <iostream>  
#include <omp.h>  
#include <vector>  
#include <ctime>  
  
const int RandomLimit = 10000;  
const bool Print = true;  
  
std::vector<std::vector<int>> create\_random\_matrix(int);  
  
std::vector<std::vector<int>> create\_empty\_matrix(int);  
  
void print\_matrix(const std::vector<std::vector<int>> &);  
  
int main() {  
 srand(time(nullptr));  
  
 int N, num\_threads;  
 double start, end, diff;  
 std::vector<std::vector<int>> matrix;  
  
 std::cout << "Enter N: ";  
 std::cin >> N;  
  
 std::cout << "Enter num of threads: ";  
 std::cin >> num\_threads;  
  
 matrix = create\_random\_matrix(N);  
  
 if (Print) {  
 print\_matrix(matrix);  
 }  
  
 start = omp\_get\_wtime();  
  
 std::vector<int> localMins;  
 for (int i = 0; i < N; ++i) {  
 localMins.push\_back(0);  
 }  
  
 int localMin = **INT32\_MAX**;  
 int max = -1;  
  
 #pragma omp parallel num\_threads(num\_threads)  
 {  
 #pragma omp for schedule(static)  
 for (int i = 0; i < N; ++i) {  
 localMin = **INT32\_MAX**;  
 for (int j = 0; j < N; ++j)  
 if (matrix[i][j] < localMin)  
 localMin = matrix[i][j];  
 localMins[i] = localMin;  
 }  
 #pragma omp flush(localMins)  
 #pragma omp for schedule(static)  
 for (int i = 0; i < N; ++i) {  
 #pragma omp critical  
 if (localMins[i] > max)  
 max = localMins[i];  
 }  
 }  
  
 if (Print) {  
 for (int min : localMins)  
 std::cout << min << " ";  
 std::cout << std::endl;  
 std::cout << std::endl;  
 }  
  
 end = omp\_get\_wtime();  
 diff = end - start;  
 std::cout << diff << std::endl;  
 std::cout << std::endl;  
  
 std::cout << "max = " << max << std::endl;  
  
 return 0;  
}  
  
std::vector<std::vector<int>> create\_random\_matrix(int size) {  
 std::vector<std::vector<int>> matrix;  
  
 for (int i = 0; i < size; ++i) {  
 matrix.emplace\_back();  
 for (int j = 0; j < size; ++j) {  
 matrix[i].push\_back(rand() % RandomLimit);  
 }  
 }  
  
 return matrix;  
}  
  
void print\_matrix(const std::vector<std::vector<int>> &matrix) {  
 for (const auto &row: matrix) {  
 for (auto column: row) {  
 std::cout << column << " ";  
 }  
 std::cout << std::endl;  
 }  
 std::cout << std::endl;  
}