МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего профессионального образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №14

РАБОТА ЗАЩИЩЕНА С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | А. Ю. Сыщиков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2  OpenMP, Распараллеливание вычислений |
|  |
| по дисциплине: [Системы с параллельной обработкой информации](https://pro.guap.ru/inside_s#subjects/2436975) |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 1742 |  |  |  | В. А. Седов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

1. **Цель работы**

Реализовать и распараллелить с помощью OpenMP различные алгоритмы.

|  |  |  |  |
| --- | --- | --- | --- |
| № варианта | Размерность задачи | Тип элемента вектора | Алгоритм |
| 16 | 20x70 | Длинное целое | Сортировка строк матрицы по возрастанию |

1. **Текст программы**

/\*

omp\_set\_nested(1); разрешает вложенную параллельность

omp\_set\_nested(0); запрещает вложенную параллельность (по умолчанию)

omp\_set\_num\_threads(2); разрешает использовать только 2 процесса в последуюзих областях

omp\_set\_dynamic(1); разрешает динамическое распределение (автоматическое), запуститься максимум ветвей (16)

omp\_set\_dynamic(0); запрещает динамическое распределение (автоматическое), будет запускаться количество ветвей оределенных пользователей, по умолчанию 1

omp\_get\_thread\_num(); возвращает номер нити

omp\_get\_num\_threads() возвращает Всего нитей в области

omp\_get\_max\_threads() возвращает максимально допустимое число нитей для использования в следующей параллельной области

omp\_get\_num\_procs возвращает количество процессоров, доступных для использования программе пользователя на момент вызова(изменчиво)

omp\_in\_parallel() возвращает 1 если она была вызвана из активной параллельной области программы, если 0 - то последовательная область

setlocale(0, "");

omp\_get\_wtime(); вернет значение времени в секундах

omp\_get\_wtick(); вернет точность таймера в секундах

#pragma omp parallel -- распаралллить Параллельная область

#pragma omp parallel reduction (+: count) -- локальная count для каждой ветки. Оператор + означате, что по завершению всей области будет соверешнно суммирование всех локальных count, поэтому по умолчанию count =0

reduction(оператор:список) – задаёт оператор и список общих пе-

ременных; для каждой переменной создаются локальные копии в каж-

дой нити; локальные копии инициализируются соответственно типу

оператора (для аддитивных операций – 0 или его аналоги, для мульти-

пликативных операций – 1 или её аналоги); над локальными копиями

переменных после выполнения всех операторов параллельной области

выполняется заданный оператор; оператор это: для языка Си – +, \*, -,

&, |, ^, &&, ||, для языка Фортран – +, \*, -, .and., .or., .eqv., .neqv.,

max, min, iand, ior, ieor; порядок выполнения операторов не опреде-

лён, поэтому результат может отличаться от запуска к запуску.

#pragma omp parallel num\_threads(3) разрешает в этой области использовать 3 процесса

#pragma omp master позволяет обеспечить печать только процессом-мастером

#pragma omp single Одна нить будет выполнять данный фрагмент, а все остальные нити будут ожидать завершения её работы

#pragma omp single nowait Одна нить будет выполнять данный фрагмент, а все остальные следующие действия, не дожидаясь ее завершения

#сложнаpragma omp single copyprivate(n) - после выпонения нити, новое значение n будет использоваться во всех нитях до и после в нашей области.

#pragma omp master выделяют участок кода, который будет выполнен только нитью-мастером. Синхронизации нет.

shared - общие переменные

private - локальные данные

shared - данные перед входом в область

private - порожденные в области

Исключение составляют

переменные, являющиеся счетчиками итераций в цикле, по очевидным при-

чинам

firstprivate - создается лок переменная, инициализация автоматическая - предыдущее значение, из общего потока

private - создается лок переменная, инициализация не происходит.

#pragma omp for

Итеративная переменная неявно

делается локальной при входе в цикл. После завершения цикла значение ите-

ративной переменной цикла не определено, если она не указана в опции

lastprivate.

\*/

#include <omp.h>

#include <time.h>

#include <locale.h>

#include <iostream>

using namespace std;

constexpr auto N = 6, M =9;

// Функция быстрой сортировки

void quickSort(long int\* numbers, long int left, long int right)

{

int pivot; // разрешающий элемент

int l\_hold = left; //левая граница

int r\_hold = right; // правая граница

pivot = numbers[left];

while (left < right) // пока границы не сомкнутся

{

while ((numbers[right] >= pivot) && (left < right))

right--; // сдвигаем правую границу пока элемент [right] больше [pivot]

if (left != right) // если границы не сомкнулись

{

numbers[left] = numbers[right]; // перемещаем элемент [right] на место разрешающего

left++; // сдвигаем левую границу вправо

}

while ((numbers[left] <= pivot) && (left < right))

left++; // сдвигаем левую границу пока элемент [left] меньше [pivot]

if (left != right) // если границы не сомкнулись

{

numbers[right] = numbers[left]; // перемещаем элемент [left] на место [right]

right--; // сдвигаем правую границу вправо

}

}

numbers[left] = pivot; // ставим разрешающий элемент на место

pivot = left;

left = l\_hold;

right = r\_hold;

if (left < pivot) // Рекурсивно вызываем сортировку для левой и правой части массива

quickSort(numbers, left, pivot - 1);

if (right > pivot)

quickSort(numbers, pivot + 1, right);

}

/\*int main(int argc, char\* argv[])

{

setlocale(0, "");

unsigned int A[N], B[N], C[N];

int i, n;

for (i = 0; i < N; i++) { A[i] = 0; B[i] = rand() % 10; C[i] = rand() % 10; }

cout <<"A:";

for (i = 0; i < N; i++) { cout<< " " << A[i];}

cout << endl;

cout << "B:";

for (i = 0; i < N; i++) { cout << " " << B[i]; }

cout << endl;

cout << "C:";

for (i = 0; i < N; i++) { cout << " " << C[i]; }

cout << endl;

#pragma omp parallel shared(A, B, C) private(i, n)

{

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

#pragma omp for

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

{

A[i] = B[i] + C[i];

printf("Нить %d сложила элемент B[%d] с элементом С[%d] и записала результат %u в элемент A[%d]\n", n, i, i, A[i], i);

}

}

cout << "A:";

for (i = 0; i < N; i++) { cout << " " << A[i]; }

}\*/

int main(int argc, char\* argv[])

{

setlocale(0, "");

long int A[N][M];

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

for (int j = 0; j < M; j++) {

A[i][j] = rand() % 100;

}

}

cout << "A:";

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

for (int j = 0; j < M; j++) {

cout << " " << A[i][j];

}

cout << "\n ";

}

cout << endl;

int i, j,n,k;

#pragma omp parallel shared(A) private(i, n, j, k)

{

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

#pragma omp for

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

{

printf("Нить %d сортирует строку %d\n", n, i);

/\*for (k = 0; k < M - 1; k++) {

for (j = 0; j < M - 1 - k; j++) {

if (A[i][j] > A[i][j+1])

{

printf("\tНить %d сравнила элементы [%d][%d] и [%d][%d] и поменяла их местами\n", n, i, j, i , j + 1);

long int temp = A[i][j];

A[i][j] = A[i][j+1];

A[i][j+1] = temp;

}

else {

printf("\t\tНить %d сравнивает элементы [%d][%d] и [%d][%d] и не поменяла их.\n", n, i, j, i, j + 1);

}

}

}\*/

quickSort(A[i], 0, N - 1); // вызов функции сортировки

}

}

cout << "A:";

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

for (int j = 0; j < M; j++) {

cout << " " << A[i][j];

}

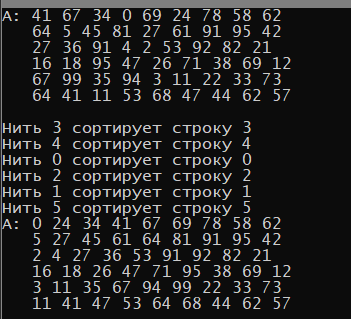
cout << "\n ";

}

cout << endl;

}

**Результат работы программы**



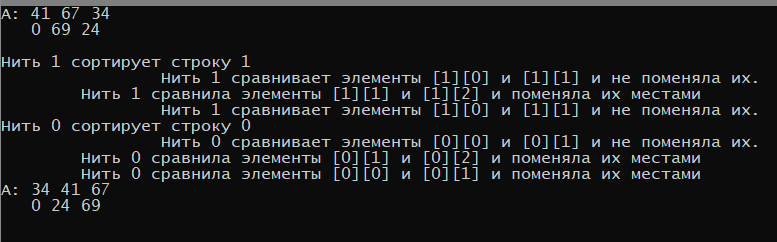


Рисунок 1. Результат работы программы.