Министерство науки и высшего образования Российской Федерации
Новосибирский Государственный технический университет
Кафедра автоматизированных систем управления

Отчет по лабораторной работе №2

по дисциплине «Параллельное программирование»

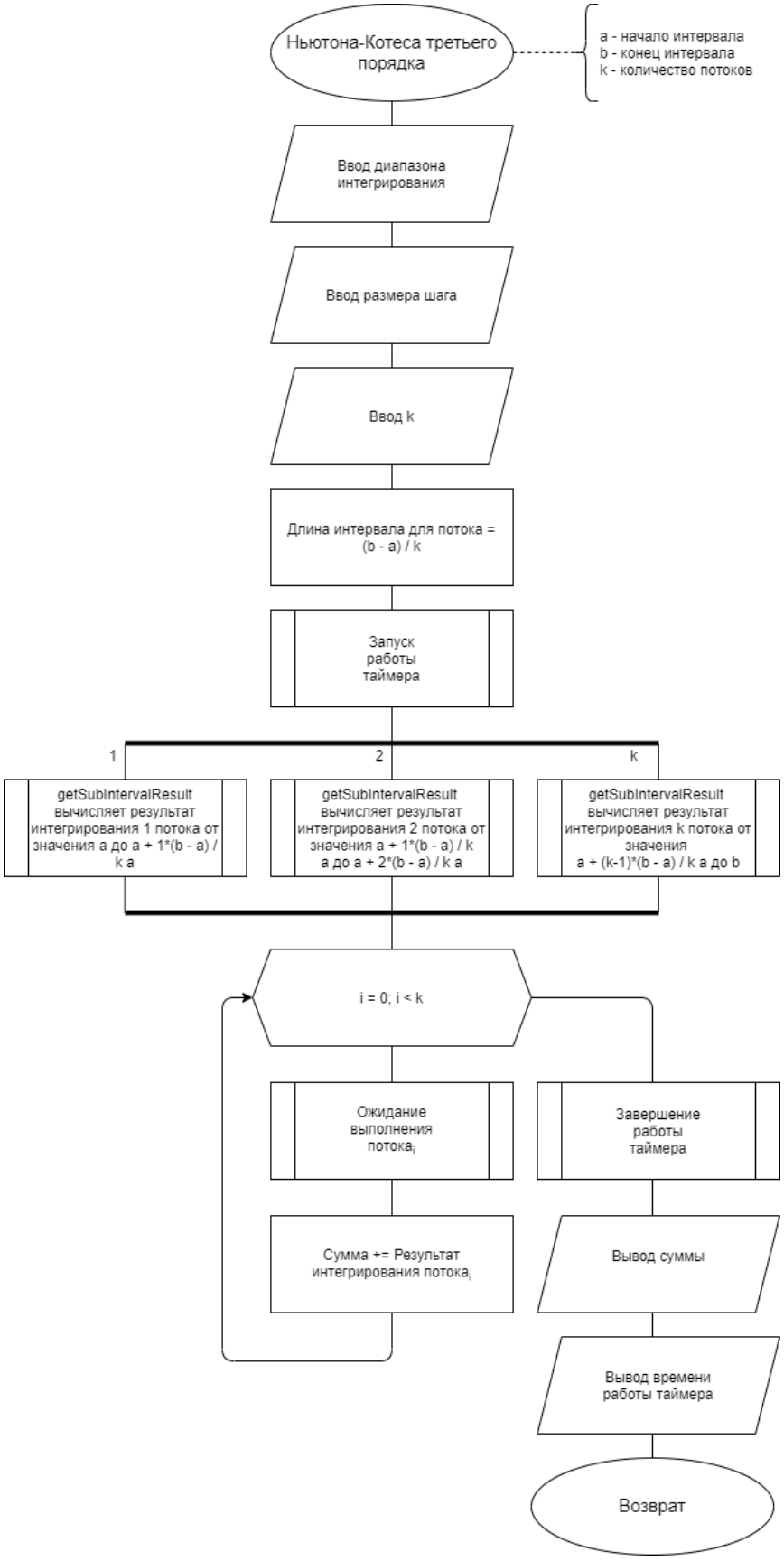
«Реализовать заданный метод численного интегрирования на языке C++ с использованием стандарта OpenMP.»

Выполнили
студенты группы АВТ-813:
Кинчаров Данил
Пайхаев Алексей
Чернаков Кирилл
Преподаватель:
Ландовский Владимир Владимирович,
к.т.н., доцент кафедры АСУ

г. Новосибирск
2020 г.

Содержание

1. Описание алгоритма.



2. Текст программы.

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "pthread.h"
#include <conio.h>
#include <string>
#include <chrono>
#include <vector>
#include <omp.h>

using namespace std;
void scanDoubleWithMessage(double*, string);
void scanThreadsNumber(int*);
int getIntervalsNumber(double, double, double);
double getThirdOrderSum(double, double);
void* getSubIntervalResult(void*);
double integrationFunction(double);
double thirdOrderNewtonCotesIntegral(int, double, double, double);
void driver();

int main()
{
	do {
		driver();
	} while (_getch() != EOF);

}

void driver() {
	int threadsNumber = 1;
	double left = 0, right = 0, step = 0;

	scanDoubleWithMessage(&left, "Enter left integration limit: ");
	scanDoubleWithMessage(&right, "Enter right integration limit: ");
	scanDoubleWithMessage(&step, "Enter integration step: ");
	scanThreadsNumber(&threadsNumber);

	auto start = std::chrono::system_clock::now();
	cout << "Result: " << thirdOrderNewtonCotesIntegral(threadsNumber, left, right,
step) << endl;
	auto end = std::chrono::system_clock::now();
	cout << "Time: " << std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count() << endl;
}

double thirdOrderNewtonCotesIntegral(int threadsNumber, double left, double right, double
step) {
	double result = 0;
	int intervalsNumber = 0, thirdOrderCn = 8, subIntervalSize = 0;
	vector<pthread_t> threads = {};
	intervalsNumber = getIntervalsNumber(left, right, step);

	#pragma omp parallel for num_threads(threadsNumber) reduction(+:result)
	for (int i = 0; i < intervalsNumber; i++) {
		result += getThirdOrderSum(left + step * i, step / 3);
	}

	result *= step / thirdOrderCn;
	return result;
}
```

```
double getThirdOrderSum(double left, double step) {
	vector<int> thirdOrderTable = { 1, 3, 3, 1 };
	double sum = 0;
	for (int i = 0; i < thirdOrderTable.size(); i++) {
		sum += integrationFunction(left + i * step) * thirdOrderTable[i];
	}
	return sum;
}

int getIntervalsNumber(double left, double right, double step) {
	int intervalsNumber = ceil(abs(right - left) / step);
	return intervalsNumber;
}

double integrationFunction( double x ) {
	return 1 / (sqrt(pow(x, 3) + 1));
}

void scanDoubleWithMessage(double* number, string message) {
	cout << message;
	cin >> *number;
}

void scanThreadsNumber(int* threadsNumber) {
	cout << "Enter threads number from 1 to 8: ";
	cin >> *threadsNumber;
	if (*threadsNumber < 1 || *threadsNumber > 8) {
		*threadsNumber = 1;
	}
}
```

3. Примеры работы программы.

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 30
Result: 2.60436
Time: 21452
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 25
Result: 2.60436
Time: 21315
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 20
Result: 2.60436
Time: 21573
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 15
Result: 2.60436
Time: 21317
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 10
Result: 2.60436
Time: 21360
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 8
Result: 2.60436
Time: 13652
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 7
Result: 2.60436
Time: 13703
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 6
Result: 2.60436
Time: 13248
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 5
Result: 2.60436
Time: 12893
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 4
Result: 2.60436
Time: 12675
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 3
Result: 2.60436
Time: 12599
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 2
Result: 2.60436
Time: 14655
```

```
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 1
Result: 2.60436
Time: 22056
```

4. Результаты экспериментов.

Подынтегральная функция:

$$f(x) = \frac{1}{\sqrt{x^3 + 1}}$$

Пределы интегрирования:

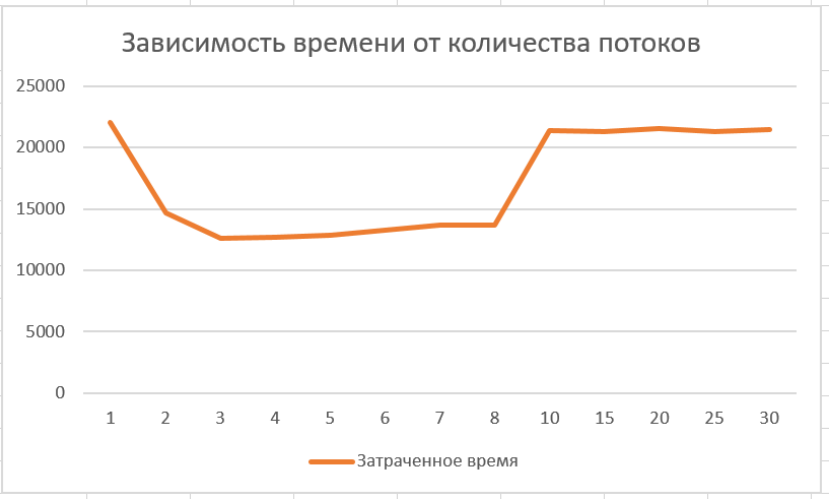
$$a = 0, b = 100$$

Шаг интегрирования:

$$1 \text{ x } 10^{-5}$$

В таблице приведены результаты выполнения программы, выполнявшейся с использованием процессора 4/4, а также график

Количество потоков	Затраченное время	Сумма
1	22056	2.60436
2	14655	2.60436
3	12599	2.60436
4	12675	2.60436
5	12893	2.60436
6	13248	2.60436
7	13703	2.60436
8	13652	2.60436
10	21360	2.60436
15	21317	2.60436
20	21573	2.60436
25	21315	2.60436
30	21452	2.60436



5. Выводы.

Основная идея "инкрементального распараллеливания" OpenMP идеально подходит для быстрого распараллеливания вычислительной программы с циклами с большим количеством независимых итераций.

При увеличении количества потоков время работы программы уменьшается. Потоки дают прирост вычислительной мощности только тогда, когда процессор может параллельно работать с всеми этими потоками. Исходя из таблицы и графика можно сказать, что на процессоре с 4 потоками время выполнения уменьшалось, пока количество потоков в программе не достигло 4. Если и дальше увеличивать число потоков, то будет наблюдаться увеличение времени выполнения из-за затрат на их создание и переключение между ними.