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

Отчет по лабораторной работе №1

по дисциплине «Параллельное программирование»

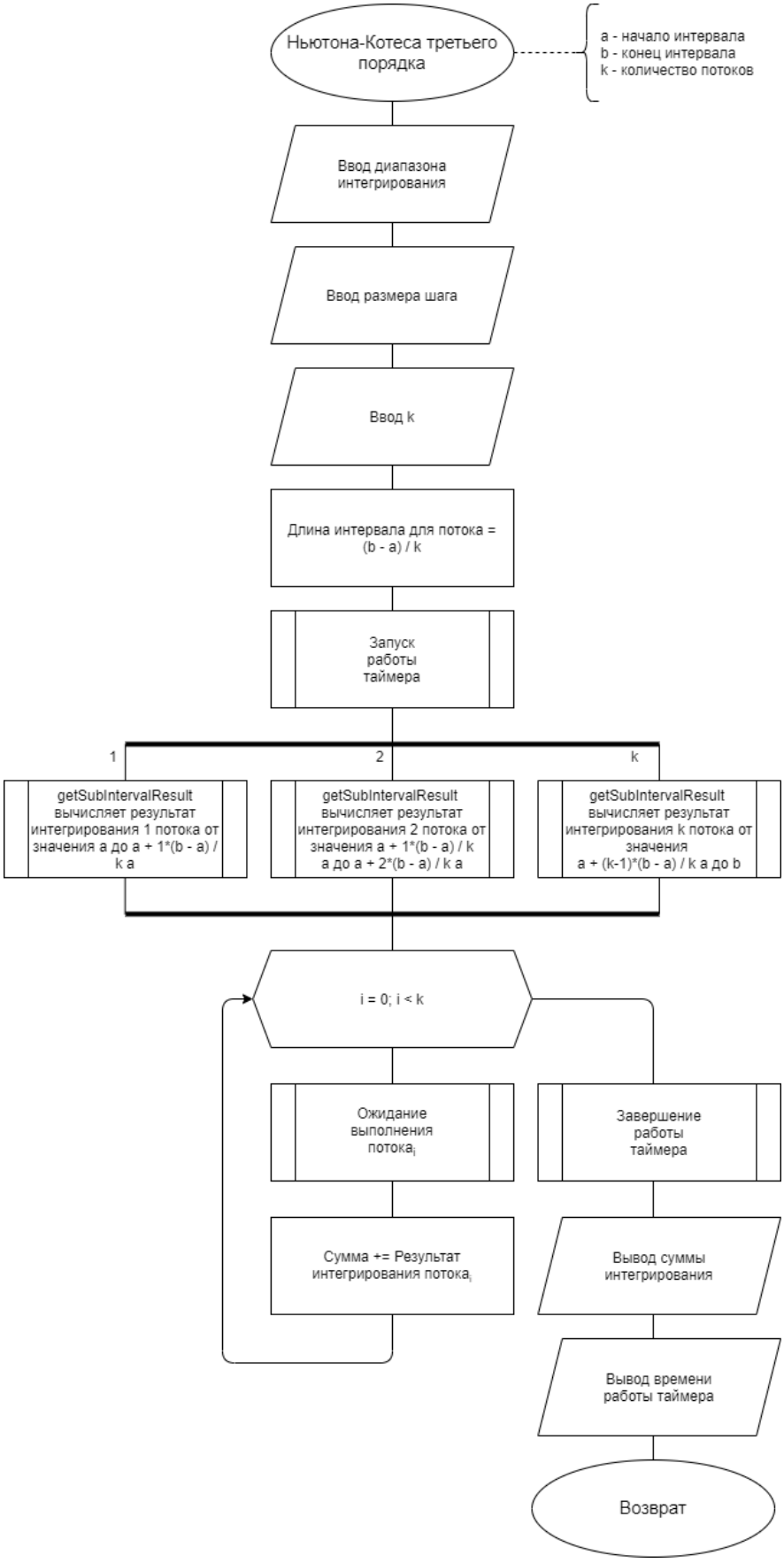
«Реализовать заданный метод численного интегрирования на языке C++ с использованием стандарта POSIX.»

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.

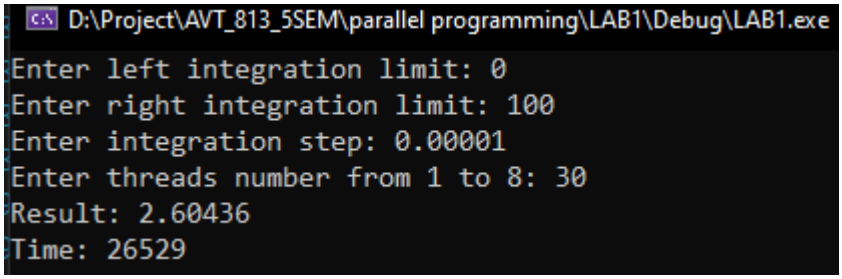


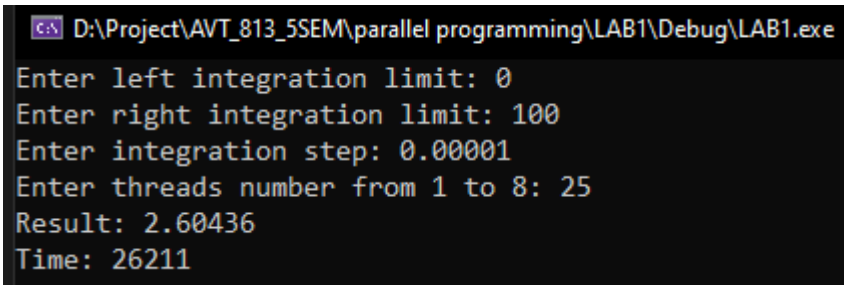
2. Текст программы.

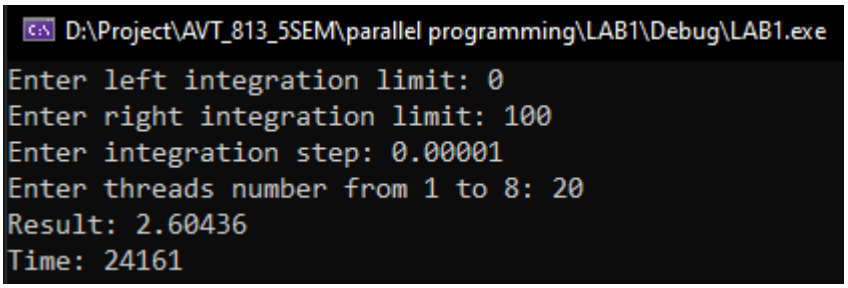
```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "pthread.h"
#include <conio.h>
#include <string>
#include <chrono>
#include <vector>

using namespace std;
#pragma comment(lib, "pthreadVCE2.lib")
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();

struct arg_struct {
	double step;
	double left;
	int subIntervalNumber;
	int subIntervalSize;
};

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);
	subIntervalSize = ceil(intervalsNumber / threadsNumber);

	for (int i = 0; i < threadsNumber; i++) {
```

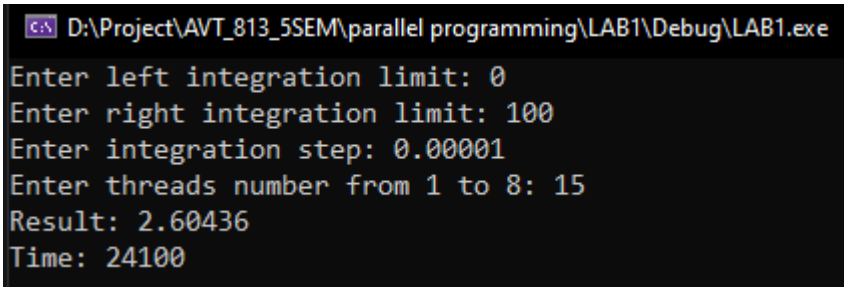
```
			struct arg_struct* args = new arg_struct();
			args->subIntervalNumber = i;
			args->subIntervalSize = subIntervalSize;
			args->step = step;
			args->left = left;
			threads.push_back(pthread_t());
			pthread_create(&threads[i], NULL, getSubIntervalResult, (void*)args);
		}
		for (int i = 0; i < threads.size(); i++) {
			double* subResult = new double(0);
			pthread_join(threads[i], (void**)&subResult);
			result += *subResult;
		}

		result *= step / thirdOrderCn;
		return result;
}

void* getSubIntervalResult(void* arguments) {
		struct arg_struct* args = (struct arg_struct*)arguments;
		double* subResult = new double(0);
		for (int i = 0; i < args->subIntervalSize; i++) {
			*subResult += getThirdOrderSum(args->left + (args->subIntervalNumber *
args->subIntervalSize + i) * args->step, args->step / 3);
		}
		return subResult;
}

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;
		}
}
```

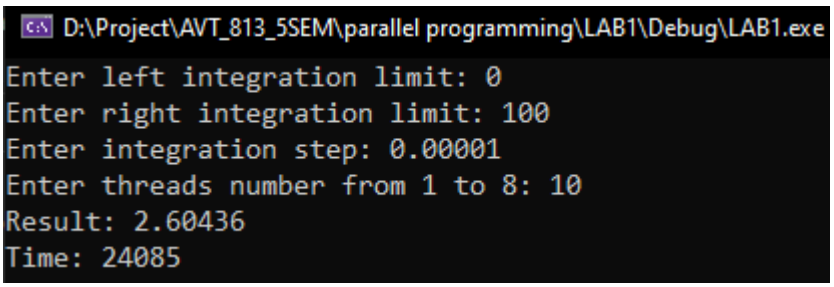
3. Примеры работы программы.

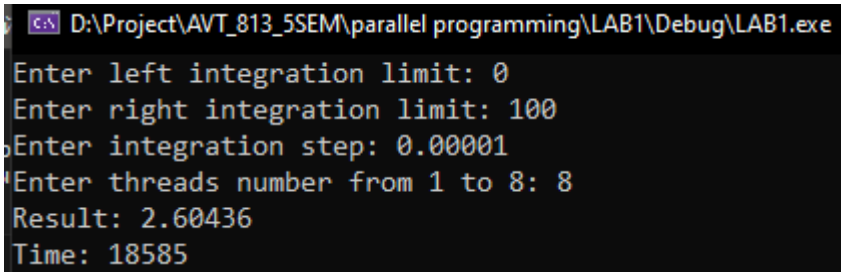


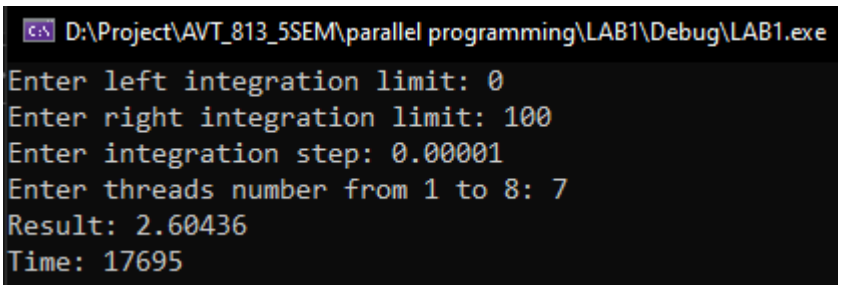


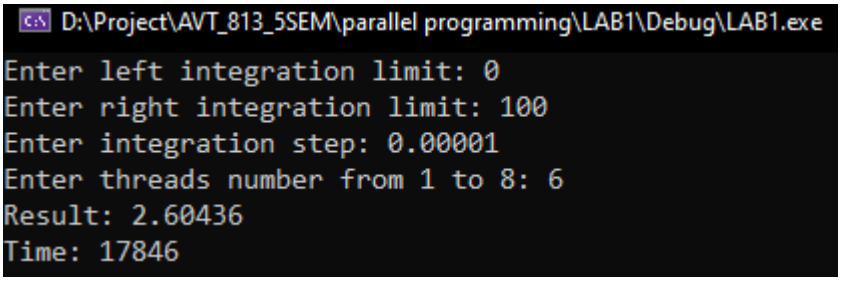


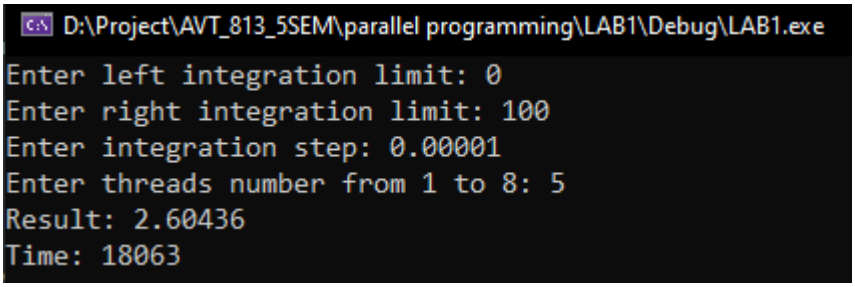


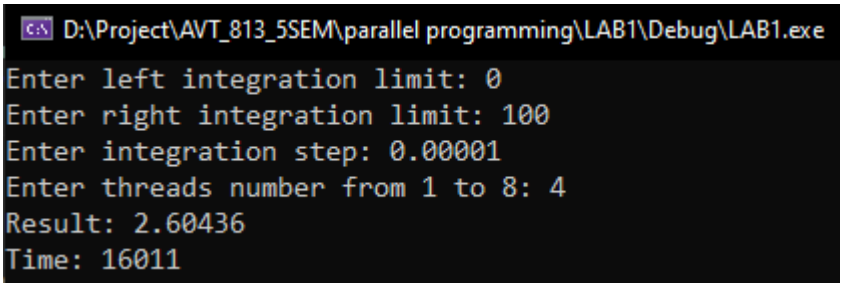


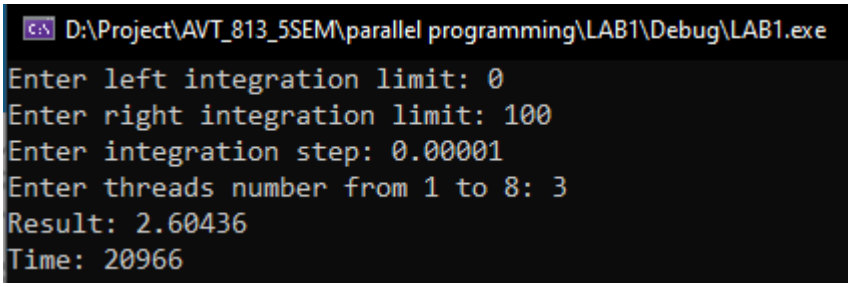


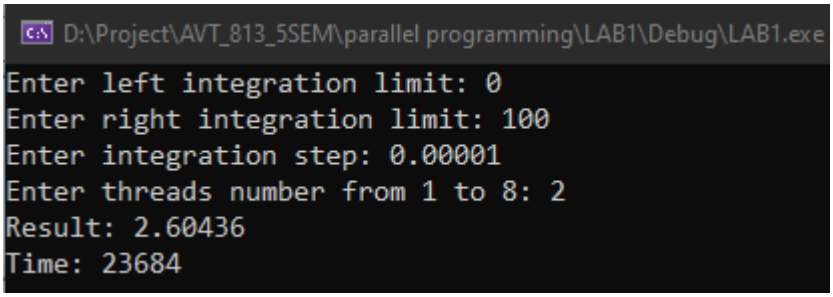

D:\Project\AVT_813_5SEM\parallel programming\LAB1\Debug\LAB1.exe
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 7
Result: 2.60436
Time: 17695

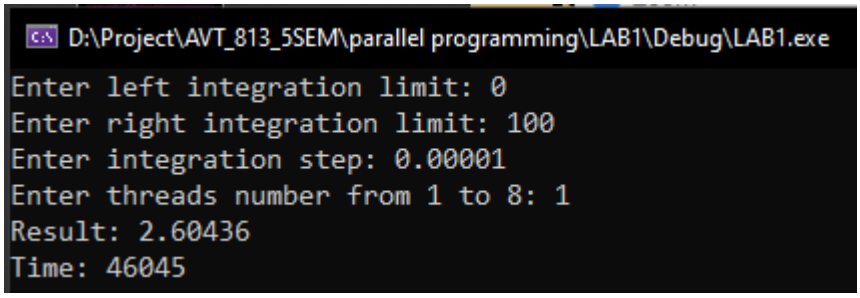

D:\Project\AVT_813_5SEM\parallel programming\LAB1\Debug\LAB1.exe
Enter left integration limit: 0
Enter right integration limit: 100
Enter integration step: 0.00001
Enter threads number from 1 to 8: 6
Result: 2.60436
Time: 17846


D:\Project\AVT_813_5SEM\parallel programming\LAB1\Debug\LAB1.exe
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: 18063


D:\Project\AVT_813_5SEM\parallel programming\LAB1\Debug\LAB1.exe
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: 16011


D:\Project\AVT_813_5SEM\parallel programming\LAB1\Debug\LAB1.exe
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: 20966


D:\Project\AVT_813_5SEM\parallel programming\LAB1\Debug\LAB1.exe
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: 23684



4. Результаты экспериментов.

Подынтегральная функция:

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

Пределы интегрирования:

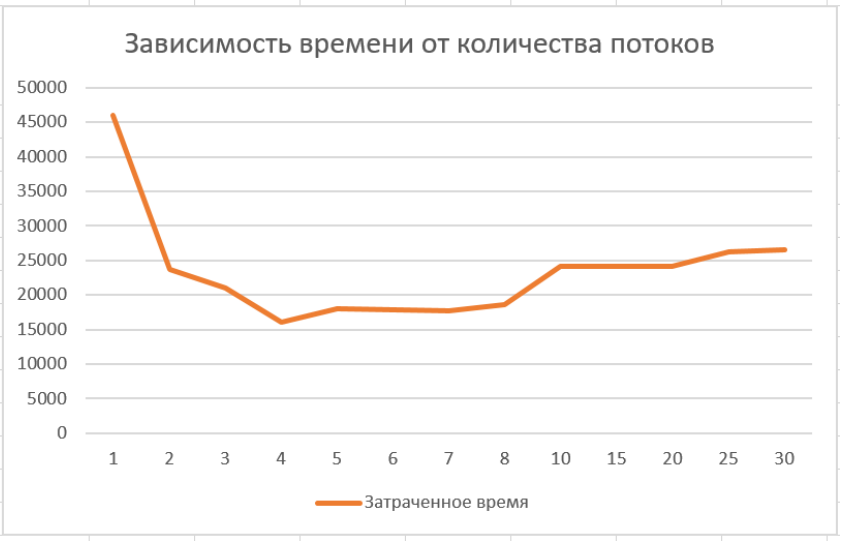
$$a = 0, b = 100$$

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	46045	2.60436
2	23684	2.60436
3	20966	2.60436
4	16011	2.60436
5	18063	2.60436
6	17846	2.60436
7	17695	2.60436
8	18585	2.60436
10	24085	2.60436
15	24100	2.60436
20	24161	2.60436
25	26211	2.60436
30	26529	2.60436



5. Выводы.

Параллельное программирование способно серьезно улучшить скорость выполнения для некоторых видов программ. Если вы хотите, чтобы программа выполнялась быстрее, разбейте ее на части и запустите каждую часть на отдельном процессоре. Параллельность является основным инструментом многопроцессорного программирования.

В наши дни, когда закон Мура постепенно перестает действовать (по крайней мере для традиционных микросхем), ускорение проявляется в форме многоядерных процессоров, нежели в ускорении отдельных чипов. Чтобы ваша программа работала быстрее, вы должны научиться эффективно использовать дополнительные процессоры, и это одна из возможностей, которую параллельное программирование может вам предоставить.

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.