Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе №1

По дисциплине «Параллельные вычисления»

«Разработка программ с использованием pthreadи OpenMPв языке C++»

Работу выполнили студенты группы №13541/3 Шаляпин Н.С._____

Работу принял преподаватель Стручков И. В._____

Санкт-Петербург

2017

Цель работы

Научиться создавать программы с использованием многопоточных технологий. Познакомиться с работой библиотек pthread и OpenMPдля языка C++. Проанализировать прирост производительности при использовании многопоточных библиотек.

Постановка задачи

Задача: Определить вероятность появления 3-грамм в тексте на русском языке.
Решить следующую задачу тремя способами:

1. Однопоточной программой.
2. Программой с использованием библиотеки pthread.
3. Программой с использованием библиотеки OpenMP.

Обзор задачи

Процессор Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz, 2501 МГц, ядер: 2, логических процессоров: 4 (*hyper-threading*)

Определители n-го порядка.

Определителем или детерминантом n-го порядка называется число записываемое в виде

$$\Delta = |a_{ik}| = \begin{vmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{vmatrix} \quad (0)$$

и вычисляемым по данным числам a_{ik} (действительным или комплексным) — элементам определителя – по следующему закону: Δ есть сумма

$$\Delta = \sum_{i} (-1)^{t(j)} a_{1j_1} a_{2j_2} \dots a_{nj_n} \quad , (1)$$

Для вычисления определителя воспользуемся его свойством:

Вычеркнем из определителя (9) n-го порядка i-ю строку и k-й столбец. Оставшееся выражение порождает определитель $(n-1)$-го порядка M_{ik}, называемый минором элемента a_{ik}. Величина же

$$A_{ik} = (-1)^{i+k} M_{ik} \quad (2)$$

называется алгебраическим дополнением или адъюнктом элемента a_{ik}.

Свойство: Сумма произведений элементов a_{ik} некоторой строки (столбца) определителя на алгебраические дополнения этих элементов равна величине определителя:

$$\Delta = \sum_{k=1}^{n} a_{ik} A_{ik} \qquad (i = 1, \ldots, n) \qquad ,(3)$$

Будем вычислять определитель матрицы 10-ого порядка и за файладata.txt и для каждого алгоритма вычислим время выполнения вычисления такого определителя для дальнейшего сравнения между собой.

data.txt

```
10
6 9 8 7 9 2 0 7 9 1
0 1 5 6 4 3 8 4 4 5
0 0 1 7 2 6 5 0 7 4
0 0 0 1 8 7 9 1 5 9
0 0 0 0 1 8 1 0 9 0
0 0 0 0 0 1 5 8 1 8
0 0 0 0 0 0 1 9 0 3
0 0 0 0 0 0 0 1 8 9
0 0 0 0 0 0 0 0 1 9
0 0 0 0 0 0 0 0 0 1
```

Однопоточная задача

Написанная однопоточная программа представлена в листинге 1.

Листинг 1. Однопоточная программа

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <locale>
#include <vector>
#include <ctime>
#include <math.h>
#include <stdlib.h>
#include <cstdlib>
#include <iomanip>
#include <windows.h>

using namespace std;

//Возвращаетматрицу matrix без row-ойстрокии col-тогостолбца, результатвnewMatrix
void getMatrixWithoutRowAndCol(double **matrix, int size, int row, int col, double
**newMatrix) {
	intoffsetRow = 0; //Смещение индекса строки в матрице
	intoffsetCol = 0; //Смещение индекса столбца в матрице
	for (inti = 0; i< size - 1; i++) {
		//Пропустить row-ую строку
		if (i == row) {
			offsetRow = 1; //Как только встретили строку, которую надо
пропустить, делаем смещение для исходной матрицы
		}

		offsetCol = 0; //Обнулить смещение столбца
```

```
            for (intj = 0; j<size - 1; j++) {
                  //Пропустить col-ый столбец
                  if (j == col) {
                        offsetCol = 1; //Встретили нужный столбец, проускаем его
смещением
                  }

                  newMatrix[i][j] = matrix[i + offsetRow][j + offsetCol];
            }
      }
}

//Вычисление определителя матрицы разложение по первой строке
double matrixDet(double **matrix, int size) {
      doubledet = 0;
      intdegree = 1; // (-1)^(1+j) из формулы определителя

      //Условие выхода из рекурсии
      if (size == 1) {
            returnmatrix[0][0];
      }
      //Условие выхода из рекурсии
      else if (size == 2) {
            return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
      }
      else {
            //Матрица без строки и столбца
            double **newMatrix = new double*[size - 1];
            for (inti = 0; i< size - 1; i++) {
                  newMatrix[i] = new double[size - 1];
            }

            //Раскладываем по 0-ой строке, цикл бежит по столбцам
            for (int j = 0; j < size; j++) {
                  //Удалить из матрицы i-ю строку и j-ый столбец
                  //Результат в newMatrix
                  getMatrixWithoutRowAndCol(matrix, size, 0, j, newMatrix);

                  //Рекурсивный вызов
                  //По формуле: сумма по j, (-1)^(1+j) * matrix[0][j] * minor_j (это
и есть сумма из формулы)
                  //где minor_j - дополнительный минор элемента matrix[0][j]
                  // (напомню, что минор это определитель матрицы без 0-ой строки и
j-го столбца)
                  det = det + (degree * matrix[0][j] * matrixDet(newMatrix, size -
1));
                  //"Накручиваем" степень множителя
                  degree = -degree;
            }

            //Чистим память на каждом шаге рекурсии(важно!)
            for (inti = 0; i< size - 1; i++) {
                  delete[] newMatrix[i];
            }
            delete[] newMatrix;
      }

      return det;
}


int _tmain(intargc, _TCHAR* argv[])
{
```

```
	ifstream in("data.txt");
	if (!in.is_open())
		return 1;
	//размерность матрицы
	int m;
	double d;
	//вводим n
	in >> m;
	printf("%d\n", m);

	//определяем вектор размером mxm
	double **mas;
	mas = new double*[m];

	for (inti = 0; i< m; i++) {
		mas[i] = new double[m];
		for (int j = 0; j < m; j++) {
			in>>mas[i][j];//считывание матрицы из файла
			//cout<<mas[i][j]<<" ";//вывод матрицы в консоль
		}
		//printf("\n");
	}
	//printf("\n");
	unsigned inttimeStart = clock();
	d = matrixDet(mas, m);
	cout<< "Determinant = " << d << "\n";

	unsigned inttimeEnd = clock();
	unsigned inttimeRezult = timeEnd - timeStart;
	cout<< "Time Work Program = " <<timeRezult<< "\n";

	for (inti = 0; i<m; i++) delete[] mas[i];
	delete[] mas;
	in.close();

	system("pause");
	return 0;
}
```

Выполнение программы:

```
10
Determinant = 6
TimeWorkProgram = 5318
Для продолжения нажмите любую клавишу . . .
```

Многопоточная программы с использованием библиотеки pthread

POSIX Threads — стандарт POSIX реализации потоков (нитей) выполнения. Стандарт POSIX.lc, Threadsextensions (IEEE Std1003,le-1995) определяет API для управления потоками, их синхронизации и планирования.

Алгоритм распараллеливания реализован следующим образом: каждому потоку на вычисление отдается по возможности равное количество слагаемых (если размерность матрицы делится без остатка на количество потоков, если при делении есть остаток – последнему потоку отдаются остаточные слагаемые) для частичного вычисления детерминанта матрицы (формула 3).

Код многопоточной программы с использованием библиотеки pthreadпредставлен в листинге 2.

Листинг 2. Многопоточная программы с использованием библиотеки pthread.

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <locale>
#include <vector>
#include <ctime>
#include <cmath>
#include <thread>
#include <mutex>
#include <math.h>
#include <stdlib.h>
#include <cstdlib>
#include <iomanip>
#include <windows.h>

using namespace std;

std::mutexg_lock;

//Возвращает матрицу matrix без row-ой строки и col-того столбца, результат в newMatrix
void getMatrixWithoutRowAndCol(double **matrix, int size, int row, int col, double
**newMatrix) {
	intoffsetRow = 0; //Смещение индекса строки в матрице
	intoffsetCol = 0; //Смещение индекса столбца в матрице
	for (inti = 0; i< size - 1; i++) {
		//Пропустить row-ую строку
		if (i == row) {
			offsetRow = 1; //Как только встретили строку, которую надо
пропустить, делаем смещение для исходной матрицы
		}

		offsetCol = 0; //Обнулить смещение столбца
		for (intj = 0; j<size - 1; j++) {
			//Пропустить col-ый столбец
			if (j == col) {
				offsetCol = 1; //Встретили нужный столбец, проускаем его
смещением
			}

			newMatrix[i][j] = matrix[i + offsetRow][j + offsetCol];
		}
	}
}

//Вычисление определителя матрицы разложение по первой строке
double matrixDet(double **matrix, int size) {
	doubledet = 0;
	intdegree = 1; // (-1)^(1+j) из формулы определителя

	//Условие выхода из рекурсии
	if (size == 1) {
		returnmatrix[0][0];
	}
	//Условие выхода из рекурсии
	else if (size == 2) {
		return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
	}
	else {
		//Матрица без строки и столбца
		double **newMatrix = new double*[size - 1];
		for (inti = 0; i< size - 1; i++) {
			newMatrix[i] = new double[size - 1];
		}
```

```
		//Раскладываем по 0-ой строке, цикл бежит по столбцам
		for (int j = 0; j < size; j++) {
			//Удалить из матрицы i-ю строку и j-ый столбец
			//Результат в newMatrix
			getMatrixWithoutRowAndCol(matrix, size, 0, j, newMatrix);

			//Рекурсивный вызов
			//По формуле: сумма по j, (-1)^(1+j) * matrix[0][j] * minor_j (это
и есть сумма из формулы)
			//где minor_j - дополнительный минор элемента matrix[0][j]
			// (напомню, что минор это определитель матрицы без 0-ой строки и
j-го столбца)
			det = det + (degree * matrix[0][j] * matrixDet(newMatrix, size -
1));
			//"Накручиваем" степень множителя
			degree = -degree;
		}

		//Чистим память на каждом шаге рекурсии(важно!)
		for (inti = 0; i< size - 1; i++) {
			delete[] newMatrix[i];
		}
		delete[] newMatrix;
	}

	return det;
}
void treadFunc(double **matrix, int size, double &det, int start, int end) {

	doubledetreg;
	//intdegree = 1; // (-1)^(1+j) из формулы определителя
	//Матрицабезстрокиистолбца

	double **newMatrix = new double*[size - 1];
	for (inti = 0; i< size - 1; i++) {
		newMatrix[i] = new double[size - 1];
	}

	for (start; start < end; start++) {
		getMatrixWithoutRowAndCol(matrix, size, 0, start, newMatrix);
		detreg = matrixDet(newMatrix, size - 1);
		det = det + pow((-1), start) * matrix[0][start] * detreg;
		//degree = -degree;
	}
	for (inti = 0; i< size - 1; i++) {
		delete[] newMatrix[i];
	}
	delete[] newMatrix;
}

int _tmain(intargc, _TCHAR* argv[])
{


	ifstream in("data1.txt");
	if (!in.is_open())
		return 1;
	//размерность матрицы
	int m;
	intnumberOfProcesses = 4;
	double *dett = new double[numberOfProcesses];

	double summ = 0;
	double det = 0;
```

```
	double det1 = 0;
	double det2 = 0;
	double det3 = 0;
	double det4 = 0;
	int step = 0;
	//вводим n
	in >> m;
	printf("%d\n", m);

	//определяем вектор размером mxm
	double **mas;
	mas = new double*[m];

	for (inti = 0; i< m; i++) {
		mas[i] = new double[m];
		for (int j = 0; j < m; j++) {
			in>>mas[i][j];//считывание матрицы из файла
			//cout<< mas[i][j]<<" ";//выводматрицы
		}
	}


	if (numberOfProcesses != 0)
		step = m / numberOfProcesses;


	//vector<thread*>vecThreads;

	unsigned inttimeStart = clock();
	if (m >= 3)
	{

		/*for (int k = 0; k <numberOfProcesses; k++)
		{
		if (k == numberOfProcesses - 1)
		{
		auto th = new std::thread(treadFunc, mas, m, std::ref(dett[k]), step*k,
m);
		vecThreads.push_back(th);
		cout<< step*k << " " << m << "\n";
		}
		else
		{
		auto th = new std::thread(treadFunc, mas, m, std::ref(dett[k]), step*k,
step*(k + 1) - 1);
		vecThreads.push_back(th);
		cout<< step*k << " " << step*(k + 1) - 1 << "\n";
		}
		}


		for (auto &th : vecThreads)
		{
		th->join();
		delete th;
		}
		vecThreads.clear();*/

		//treadFunc(mas, m, det1, 0, 10);
		//std::thread t1(treadFunc, mas, m, std::ref(det1), 0, 10);


		if (numberOfProcesses == 0)
		{
			treadFunc(mas, m, det1, 0, m);
```

```
		}

		if (numberOfProcesses == 1)
		{
			std::thread t1(treadFunc, mas, m, std::ref(det1), 0, m);
			t1.join();
		}
		if (numberOfProcesses == 2)
		{
			std::thread t1(treadFunc, mas, m, std::ref(det1), step * 0, step*(0
+ 1) - 1);
			std::thread t2(treadFunc, mas, m, std::ref(det2), step * 1, m);
			t1.join();
			t2.join();
		}
		if (numberOfProcesses == 3)
		{
			std::thread t1(treadFunc, mas, m, std::ref(det1), step * 0, step*(0
+ 1) - 1);
			std::thread t2(treadFunc, mas, m, std::ref(det2), step * 1, step*(1
+ 1) - 1);
			std::thread t3(treadFunc, mas, m, std::ref(det3), step * 2, m);
			t1.join();
			t2.join();
			t3.join();
		}
		if (numberOfProcesses == 4)
		{
			std::thread t1(treadFunc, mas, m, std::ref(det1), step * 0, step*(0
+ 1) - 1);
			std::thread t2(treadFunc, mas, m, std::ref(det2), step * 1, step*(1
+ 1) - 1);
			std::thread t3(treadFunc, mas, m, std::ref(det3), step * 2, step*(2
+ 1) - 1);
			std::thread t4(treadFunc, mas, m, std::ref(det4), step * 3, m);
			t1.join();
			t2.join();
			t3.join();
			t4.join();
		}

		det = det1 + det2 + det3 + det4;

		/*for (inti = 0; i<numberOfProcesses; i++)
			summ = summ + dett[i];*/
	}
	else
	det = matrixDet(mas, m);


	unsigned inttimeEnd = clock();

	cout<< "Determinant = " <<det<< "\n";

	unsigned inttimeRezult = timeEnd - timeStart;
	cout<< "Time Work Program = " <<timeRezult<< "\n";

	for (inti = 0; i<m; i++) delete[] mas[i];
	delete[] mas;
	delete[] dett;
	in.close();

	system("pause");
	return 0;
}
```

Выполнение программы:

```
10
Determinant = 6
TimeWorkProgram = 2811
Для продолжения нажмите любую клавишу . . .
```

Многопоточная программа с использованием библиотеки MPI

Message Passing Interface (MPI, интерфейс передачи сообщений) — программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. Базовым механизмом связи между MPI процессами является передача и приём сообщений. Сообщение несёт в себе передаваемые данные и информацию, позволяющую принимающей стороне осуществлять их выборочный приём. Алгоритм распараллеливания остается прежним. Исходный код однопоточной программы был изменен следующим образом:

Код многопоточной программы с использованием библиотеки MPI представлен в листинге 3.

Листинг 3. Многопоточная программы с использованием библиотеки MPI.

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <locale>
#include <vector>
#include <ctime>
#include <cmath>
#include <thread>
#include <mutex>
#include <math.h>
#include <stdlib.h>
#include <cstdlib>
#include <iomanip>
#include <windows.h>
#include <mpi.h>

using namespace std;

std::mutexg_lock;

//Возвращаетматрицу matrix без row-ойстрокии col-тогостолбца, результатвnewMatrix
void getMatrixWithoutRowAndCol(double **matrix, int size, int row, int col, double
**newMatrix) {
	intoffsetRow = 0; //Смещение индекса строки в матрице
	intoffsetCol = 0; //Смещение индекса столбца в матрице
	for (inti = 0; i< size - 1; i++) {
		//Пропустить row-ую строку
		if (i == row) {
			offsetRow = 1; //Как только встретили строку, которую надо
пропустить, делаем смещение для исходной матрицы
		}

		offsetCol = 0; //Обнулить смещение столбца
		for (intj = 0; j<size - 1; j++) {
			//Пропустить col-ый столбец
			if (j == col) {
				offsetCol = 1; //Встретили нужный столбец, проускаем его
```

```
смещением
                    }

                    newMatrix[i][j] = matrix[i + offsetRow][j + offsetCol];
             }
      }
}

//Вычисление определителя матрицы разложение по первой строке
double matrixDet(double **matrix, int size) {
      doubledet = 0;
      intdegree = 1; // (-1)^(1+j) из формулы определителя

      //Условие выхода из рекурсии
      if (size == 1) {
            returnmatrix[0][0];
      }
      //Условие выхода из рекурсии
      else if (size == 2) {
            return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
      }
      else {
            //Матрица без строки и столбца
            double **newMatrix = new double*[size - 1];
            for (inti = 0; i< size - 1; i++) {
                  newMatrix[i] = new double[size - 1];
            }

            //Раскладываем по 0-ой строке, цикл бежит по столбцам
            for (int j = 0; j < size; j++) {
                  //Удалить из матрицы i-ю строку и j-ый столбец
                  //Результат в newMatrix
                  getMatrixWithoutRowAndCol(matrix, size, 0, j, newMatrix);

                  //Рекурсивный вызов
                  //По формуле: сумма по j, (-1)^(1+j) * matrix[0][j] * minor_j (это
и есть сумма из формулы)
                  //где minor_j - дополнительный минор элемента matrix[0][j]
                  // (напомню, что минор это определитель матрицы без 0-ой строки и
j-го столбца)
                  det = det + (degree * matrix[0][j] * matrixDet(newMatrix, size -
1));
                  //"Накручиваем" степень множителя
                  degree = -degree;
            }

            //Чистим память на каждом шаге рекурсии(важно!)
            for (inti = 0; i< size - 1; i++) {
                  delete[] newMatrix[i];
            }
            delete[] newMatrix;
      }

      return det;
}


int _tmain(intargc, char* argv[])
{


      ifstream in("data1.txt");
      if (!in.is_open())
            return 1;
      //размерность матрицы
```

```
	int m;

	double det = 0;
	int step = 0;
	double summ = 0;

	//вводим n
	in >> m;
	printf("%d\n", m);

	//определяем вектор размером mxm
	double **mas;
	mas = new double*[m];

	for (inti = 0; i< m; i++) {
		mas[i] = new double[m];
		for (int j = 0; j < m; j++) {
			in>>mas[i][j];//считывание матрицы из файла
			//cout<< mas[i][j]<<" ";//выводматрицы
		}
	}

	unsigned inttimeStart = clock();

	if (m >= 3)
	{

		intmyid, numprocs = 5;

		if (numprocs != 0)
			step = m / numprocs;


		if (intrc = MPI_Init(&argc, &argv))
		{
			cout<< "Ошибка запуска, выполнение остановлено " <<endl;
			MPI_Abort(MPI_COMM_WORLD, rc);
		}

		MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
		MPI_Comm_rank(MPI_COMM_WORLD, &myid);

		MPI_Bcast(&step, 1, MPI_INT, 0, MPI_COMM_WORLD);

		int start = 0;
		int end = m;

		double detreg;

		if (myid == numprocs)
		{
			start = step*(numprocs - 1);
			end = m;
		}
		else
		{
			start = step*myid;
			end = step*(myid + 1) - 1;
		}


		double **newMatrix = new double*[m - 1];
		for (inti = 0; i< m - 1; i++) {
			newMatrix[i] = new double[m - 1];
		}
```

```
            for (start; start < end; start++) {
                  getMatrixWithoutRowAndCol(mas, m, 0, start, newMatrix);
                  detreg = matrixDet(newMatrix, m - 1);
                  det = det + pow((-1), start) * mas[0][start] * detreg;
            }
            for (inti = 0; i< m - 1; i++) {
                  delete[] newMatrix[i];
            }
            delete[] newMatrix;

            MPI_Reduce(&det, &summ, 1, MPI_LONG_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
            MPI_Finalize();

      }

      else
            summ = matrixDet(mas, m);

            unsigned inttimeEnd = clock();

            cout<< "Determinant = " <<summ<< "\n";

            unsigned inttimeRezult = timeEnd - timeStart;
            cout<< "Time Work Program = " <<timeRezult<< "\n";

            for (inti = 0; i< m; i++) delete[] mas[i];
            delete[] mas;
            in.close();

            system("pause");

      return 0;
}
```

Выполнение программы:

```
10
Determinant = 6
TimeWorkProgram = 520
Для продолжения нажмите любую клавишу . . .
```

Для работы программы в MPI необходимо перед началом межпроцессного обмена вызвать функцию *MPI_Init*, а в конце вызвать *MPI_Finalize*. На самом деле MPI запускает несколько копий одного и того же процесса, у них различаются только ранги. Ранг и количество процессов можно получить с помощью функций *MPI_Comm_rank* и *MPI_Comm_size*. Каждый процесс MPI запускает функцию *thread*,каждой из которых передается определённое количество слагаемых для определения детерминанта. Результат работы этой функции сохраняется в переменную *det*. Далее с помощью функции *MPI_Reduce* значения данной переменной объединяются в другую переменную на процессе с указанным рангом. Объединения происходит по указанной операции. В данном случае результаты объединяются на процессе с рангом 0 в переменную *summ* с помощью суммирования. Далее этот процесс выводит результаты в консоль.

Многократный запуск программ и подсчет вероятностных характеристик

Для подсчета вероятностных характеристик был создан скрипт, запускающий программу 100 раз и подсчитывающий времена ее исполнения. На основе полученных данных подсчитывается математическое ожидание, дисперсия и доверительных интервал. Скрипт представлен в листинге 3

Листинг 3. Скрипт многократного запуска программы.

```
# -*- coding: cp1251 -*-

import sys
import subprocess
from math import sqrt

# arguments
args = list(sys.argv )

programm = "C:\Users\Admin\Documents\Visual Studio
2013\Projects\ParallelsProgramProject\DeterminantRecursionPthread\Debug\
DeterminantRecursionPthread.exe"
numRepeats = 100

if len(args) >= 5:
programm = args [1]]
numRepeats = int ( args [2])


# program
PIPE = subprocess.PIPE

for threads in [1, 2, 4, 8]:#[4]:#
timeList = []
    for num in range(numRepeats):
        p = subprocess.Popen([programm, str(threads)], stdout=PIPE)
        for line in p.stdout:
            if 'runtime without reading = ' in line :
timeList.append(int(line.split()[-1]))

    m=sum(timeList)/numRepeats
disp = 0.00
    for val in timeList :
disp = disp + (val - m) ** 2

    if numRepeats == 1:
disp = disp / numRepeats
    else :
disp = disp / ( numRepeats - 1)

    sigma = sqrt(disp)

    t=1.984
interHigh = m + t*(sigma/(sqrt(numRepeats)))
interLow = m - t*(sigma/(sqrt(numRepeats)))
    print("{} threads : average = {}, dispersion = {}, interval = [{},
{}]".format(threads , m , disp, interHigh, interLow))
```

На основе выводов данного скрипта была построена сводная таблица результатов для всех программ:

Табл.1. Сводная таблица результатов для 100 запусков каждой программы.

	Число потоков	Мат. Ожид.	Дисперси	Дов. Интер. 0.95%
Simple	1	55	22.54	[54.05- 55.94]
pThread	1	59	12.21	[58.30- 59.69]
	2	48	12.98	[48.46- 49.93]
	4	44	25.20	[43.43- 45.56]
	8	**39**	18.40	[38.14- 39.85]
MPI	1	55	15.81	[54.21- 55.78]
	2	42	12.98	[41.28- 42.71]
	4	40	62.23	[38.00- 41.99]
	8	**38**	95.04	[35.36- 40.63]

Из таблицы 1 видно, что многопоточные приложения выигрывают по скорости выполнения у однопоточного. Так же видно, что программа с библиотекойpThreadпроигрывает у программы с использованием библиотеки MPI. Увеличение числа потоков до 8 не дало значительного прироста, т.к. процессор используемой системы имеет 4 логических потока.

Оптимальное количество потоков для программы с использованием библиотеки MPIоказалось равно 8. Однако для 4-ех потоков и более видно возрастание дисперсии, что говорит о том, что в данной конфигурации приложение срабатывает не всегда одинаково. Данные результаты так же, зависят от загруженности системы в конкретный момент времени, что может серьезно влиять на скорость выполнения программ.

Вывод

В данной работе были изучены основы создания параллельных приложений на C++. Были изучены библиотеки Pthread и MPI. Созданные программы были протестированы па разных наборах данных и были оценены характеристики времени работы. На основе проведенных экспериментов можно сделать вывод, что наиболее эффективным решением является решение на основе библиотеки MPI. Это достаточно ожидаемый результат, так как MPI — это стандарт, созданный специально для написания программ с высокой степенью параллелизма.

В общем, можно сказать что MPI гораздо более предпочтителен дня создания многопоточных программ, так как он обладает большим набором удобных функций и возможностей (например, широковещательная рассылка).

Программу удалось реализовать полностью независимой по данным, поэтому в работе не возникло необходимости использования средств синхронизации.