ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5  
«Оптимизация доступа к памяти»  
по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. ИП-813  
Бурдуковский Илья Александрович

Проверил: ст. преп. Кафедры ВС   
Токмашева Елизавета Ивановна

Новосибирск 2020

Содержание

[Содержание 2](#_Toc58880907)

[Запуск программы 8](#_Toc58880908)

[Результат работы 9](#_Toc58880909)

[Приложение №1 11](#_Toc58880910)

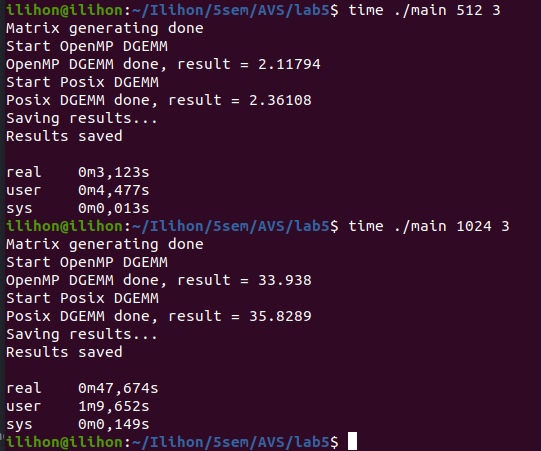
Постановка задачи

1. Для программы умножения двух квадратных матриц DGEMM BLAS разработанной в задании 4 на языке С/С++ реализовать многопоточные вычисления. В потоках необходимо реализовать инициализацию массивов случайными числами типа double и равномерно распределить вычислительную нагрузку. Обеспечить возможность задавать размерность матриц и количество потоков при запуске программы. Многопоточность реализовать несколькими способами.   
 1) С использованием библиотеки стандарта POSIX Threads.   
 2) С использованием библиотеки стандарта OpenMP.  
 3) \* C использованием библиотеки Intel TBB.  
 4) \*\* C использованием библиотеки стандарта MPI. Все матрицы помещаются в общей памяти одного вычислителя.   
 5) \*\*\* C использованием технологий многопоточности для графических сопроцессоров (GPU) - CUDA/OpenCL/OpenGL/OpenACC.   
2. Для всех способов организации многопоточности построить график зависимости коэффициента ускорения многопоточной программы от числа потоков для заданной размерностиматрицы, например, 5000, 10000 и 20000 элементов.  
3. Определить оптимальное число потоков для вашего оборудования.   
4. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

Запуск программы

Запуск программы:

Здесь указывается размер матрицы и количество потоков.



Результат работы

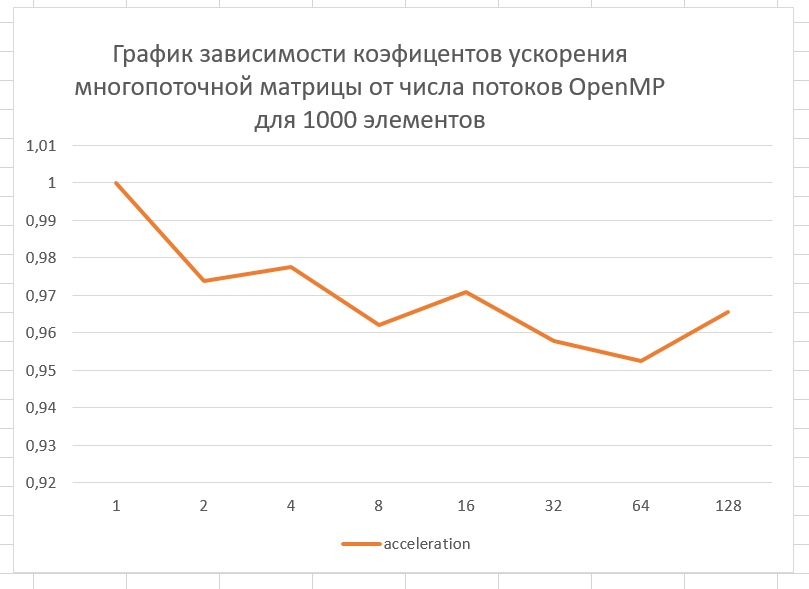


Рисунок 1. График зависимости коэффициентов ускорения многопоточной матрицы от числа потоков OpenMP для 1000 элементов.

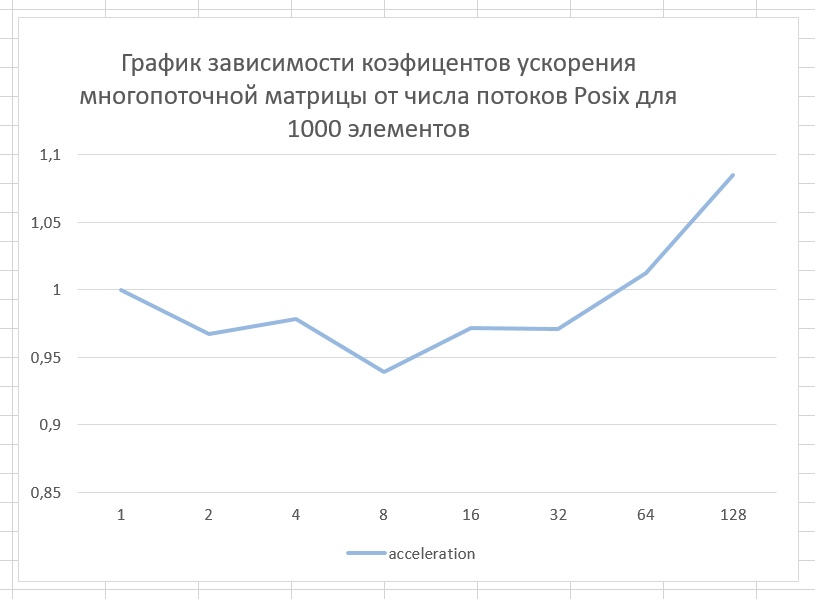


Рисунок 2. График зависимости коэффициентов ускорения многопоточной матрицы от числа потоков Posix для 1000 элементов.

**Оценить оптимальное число потков.**

На виртуальную машину выделено 3 ядра из доступных. Предполагается, что у нас имеется по потоку на ядро ввиду работы на виртуальной машине.   
Из этого делаем вывод, что наибольшая эффективность работы достигается при 3х потоках.

Приложение №1

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <fstream>

#include <time.h>

#include <omp.h>

#include <pthread.h>

using namespace std;

// g++ -o main -fopenmp -pthread main.cpp && ./main 16 4

double \*\*generate\_matrix\_random(int size) {

double \*\*m = new double \*[size];

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

m[i] = new double[size];

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

m[i][j] = (double) (rand() % 100);

}

}

return m;

}

double \*\*generate\_matrix\_incr(int size) {

double \*\*m = new double \*[size];

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

m[i] = new double[size];

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

m[i][j] = i \* size + j + 1;

}

}

return m;

}

double \*\*generate\_matrix\_zero(int size) {

double \*\*m = new double \*[size];

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

m[i] = new double[size];

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

m[i][j] = 0.0;

}

}

return m;

}

double \*\*print\_matrix(double \*\*m, int size) {

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

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

printf("%12.2f", m[i][j]);

}

printf("\n");

}

return m;

}

void openmp\_DGEMM(double \*\*a, double \*\*b, double \*\*c, int matrix\_size) {

int i, j, k;

#pragma omp parallel for private(i, j, k) shared(a, b, c)

for (i = 0; i < matrix\_size; ++i) {

for (j = 0; j < matrix\_size; ++j) {

for (k = 0; k < matrix\_size; ++k) {

c[i][j] += a[i][k] \* b[k][j];

}

}

}

}

struct Mult\_args\_t {

double \*\*a;

double \*\*b;

double \*\*c;

int size;

int left\_border;

int right\_border;

};

void \*posix\_DGEMM(void \*arg) {

Mult\_args\_t \*args = (Mult\_args\_t \*) arg;

for (int i = args->left\_border; i < args->right\_border; ++i) {

for (int j = 0; j < args->size; ++j) {

for (int k = 0; k < args->size; ++k) {

args->c[i][j] += args->a[i][k] \* args->b[k][j];

}

}

}

pthread\_exit(0);

}

void posix\_multiply\_matrix(double \*\*a, double \*\*b, double \*\*c, int matrix\_size, int threads\_count, double &result) {

int for\_one\_thread = matrix\_size / threads\_count;

unsigned int start, stop;

pthread\_t \*threads = new pthread\_t[threads\_count];

int left\_border = 0;

int right\_border;

Mult\_args\_t \*\*args = new Mult\_args\_t \*[threads\_count];

for (int i = 0; i < threads\_count; ++i) {

args[i] = new Mult\_args\_t;

right\_border = left\_border + for\_one\_thread;

args[i]->a = a;

args[i]->b = b;

args[i]->c = c;

args[i]->size = matrix\_size;

args[i]->left\_border = left\_border;

args[i]->right\_border = right\_border;

left\_border += for\_one\_thread;

}

start = clock();

for (int i = 0; i < threads\_count; ++i) {

pthread\_create(&threads[i], nullptr, posix\_DGEMM, args[i]);

}

for (int i = 0; i < threads\_count; ++i) {

pthread\_join(threads[i], nullptr);

}

stop = clock();

result = (double)(stop - start) / CLOCKS\_PER\_SEC;

}

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

int matrix\_size = 0;

int threads = 0;

if (argc != 3) {

printf("No arguments entered!\n");

exit(1);

}

matrix\_size = atoi(argv[1]);

threads = atoi(argv[2]);

double \*\*a, \*\*b, \*\*c;

a = generate\_matrix\_incr(matrix\_size);

b = generate\_matrix\_incr(matrix\_size);

c = generate\_matrix\_zero(matrix\_size);

cout << "Matrix generating done" << endl;

omp\_set\_dynamic(0);

omp\_set\_num\_threads(threads);

cout << "Start OpenMP DGEMM" << endl;

unsigned int start, stop;

double openmp\_result, posix\_result;

start = clock();

openmp\_DGEMM(a, b, c, matrix\_size);

stop = clock();

openmp\_result = (double)(stop - start) / CLOCKS\_PER\_SEC;

cout << "OpenMP DGEMM done, result = " << openmp\_result << endl;

cout << "Start Posix DGEMM" << endl;

posix\_multiply\_matrix(a, b, c, matrix\_size, threads, posix\_result);

cout << "Posix DGEMM done, result = " << posix\_result << endl;

cout << "Saving results..." << endl;

ofstream output;

output.open("result.csv", ios\_base::app);

output << "DGEM\_style;Size;Thread;Time;" << endl;

output << "OpenMP;" << matrix\_size << ";" << threads << ";" << openmp\_result << ";" << endl;

output << "Posix;" << matrix\_size << ";" << threads << ";" << posix\_result << ";" << endl;

output.close();

cout << "Results saved" << endl;

delete[] a;

delete[] b;

delete[] c;

return 0;

}

Приложение №2

Файл 2 .csv

