Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра вычислительных систем

Отчет по лабораторной работе

по дисциплине «Архитектура вычислительных систем»

Лабораторная работа №2

«Оценка производительности процессора»

Выполнил: студент 3 курса
группы ИП-811
Мироненко К. А

Проверил: доцент кафедры ВС
Ефимов А. В.

Новосибирск 2020-21 уч. год

Оглавление

1. Постановка задачи

Тема: оценка производительности процессора

Задание: р*еализовать программу для оценки производительности процессора (benchmark).*

1. Написать программу(ы) (benchmark) на языке C/C++/C# для оценки производительности процессора. В качестве набора типовых задач использовать либо минимум 3 функции выполняющих математические вычисления, либо одну функцию по работе с матрицами и векторами данных с несколькими типами данных.
2. С помощью системного таймера или с помощью процессорного регистра счетчика TSC реализовать оценку в секундах среднего времени испытания каждой типовой задачи. Оценить точность и погрешность (абсолютную и относительную) измерения времени.
3. Результаты испытаний в самой программе (или с помощью скрипта) сохранить в файл в формате CSV.
4. ***** Оценить среднее время испытания каждой типовой задачи с разным типом входных данных (целочисленные, с одинарной и двойной точностью).
5. ****** Оценить среднее время испытания каждой типовой задачи с оптимизирующими преобразования исходного кода компилятором (ключи –O1, O2, O3 и др.).
6. ******* Оценить и постараться минимизировать накладные расходы(время на вызов функций, влияние загрузки системы и т.п.) при испытании, то есть добиться максимальной точности измерений.
7. Построить сводную диаграмму производительности в зависимости от задач и выбранных исходных параметров испытаний. Оценить среднее быстродействие (производительность) для равновероятного использования типовых задач.

2. Выполнение работы

В качестве целевого языка для написания тестов производительности был выбран язык C++. В роли типовых задач были выбрана функция перемножения матриц.

За само тестирование отвечали 3 функции void DGEMM(int**, int**, int**, long long), void DGEMM(long long**, long long**, long long**, long long), void DGEMM(double**, double**, double**, long long), для каждого из типов данных соответственно.

Для получения сведений о процессоре был использован файл /proc/cpuinf.

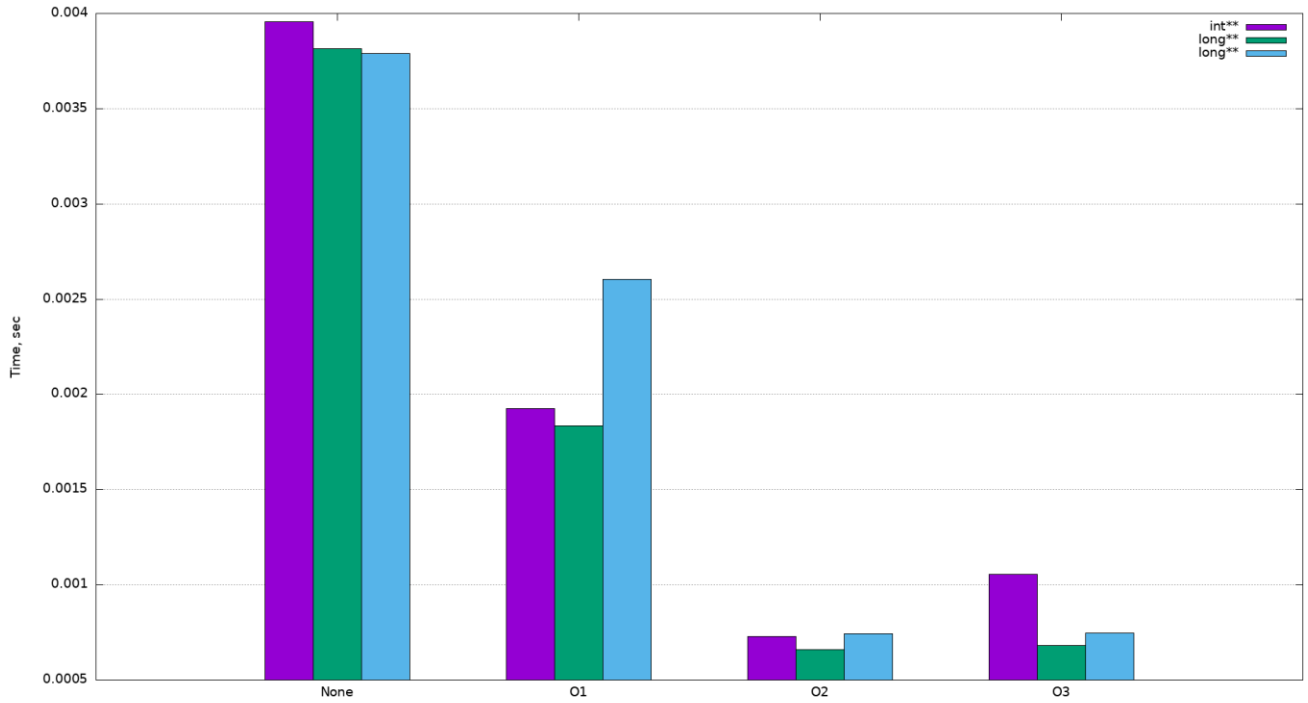
Затем все необходимые данные были сохранены в csv файл.

Чтобы оценить среднее время испытания каждой типовой задачи с разным типом входных данных и ключами оптимизации программа выполнялась несколько раз.

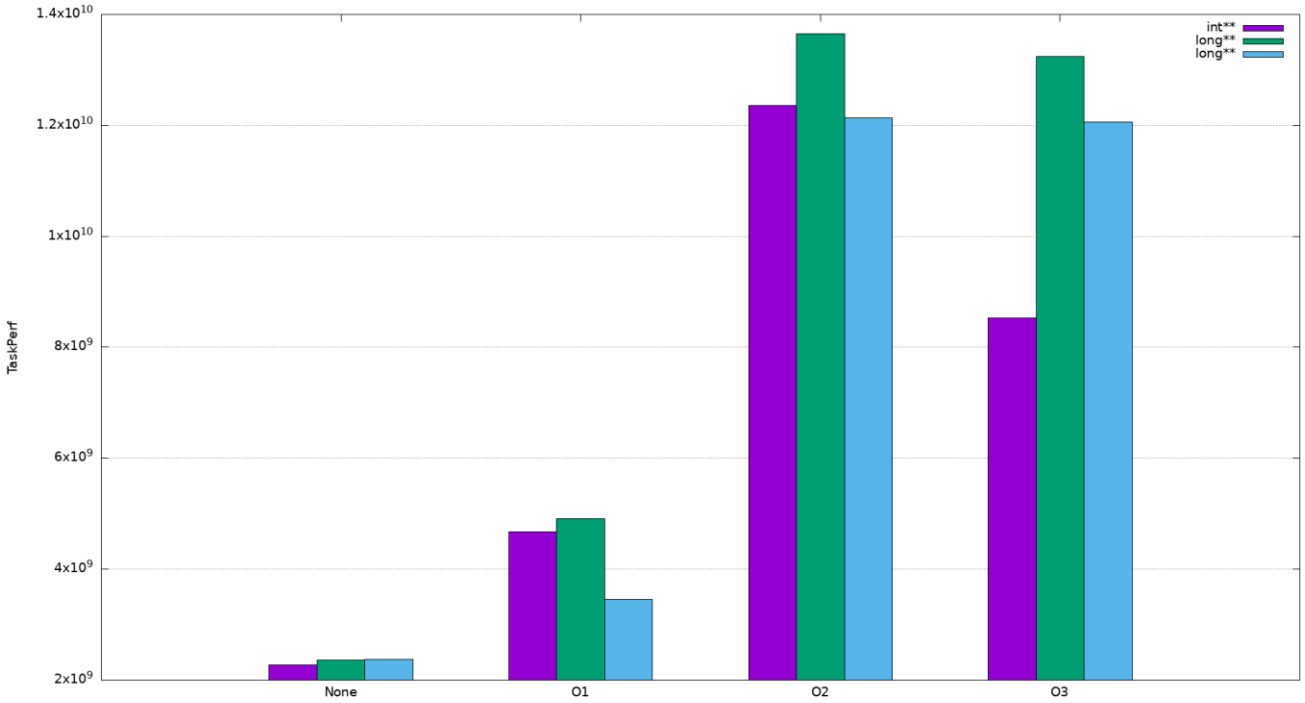
3. Примеры работы программы

	1	2	3	4	5	6	7	8	9	10	11	12
1	PModel	Task	OpType	Opt	LNum	InsCount	Timer	AvTime	AbsErr	RelErr	TaskPerf	
2	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	int**	O0	100	9000000	clock()	3.957950e-03	6.278072e-02	9.958233e-01	2.273904e+09	
3	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	long long**	O0	100	9000000	clock()	3.815950e-03	6.165510e-02	9.961742e-01	2.358521e+09	
4	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	double**	O0	100	9000000	clock()	3.789430e-03	6.144159e-02	9.962102e-01	2.375027e+09	
5	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	int**	O1	100	9000000	clock()	1.926370e-03	4.384648e-02	9.979980e-01	4.672000e+09	
6	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	long long**	O1	100	9000000	clock()	1.833670e-03	4.278208e-02	9.981656e-01	4.908190e+09	
7	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	double**	O1	100	9000000	clock()	2.602990e-03	5.095306e-02	9.973969e-01	3.457562e+09	
8	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	int**	O2	100	9000000	clock()	7.284900e-04	2.697889e-02	9.991358e-01	1.235432e+10	
9	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	long long**	O2	100	9000000	clock()	6.592200e-04	2.566681e-02	9.993407e-01	1.365250e+10	
10	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	double**	O2	100	9000000	clock()	7.419900e-04	2.722936e-02	9.992561e-01	1.212954e+10	
11	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	int**	O3	100	9000000	clock()	1.055470e-03	3.246497e-02	9.985830e-01	8.527007e+09	
12	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	long long**	O3	100	9000000	clock()	6.798700e-04	2.606541e-02	9.993170e-01	1.323782e+10	
13	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx	matrixMultiplication	double**	O3	100	9000000	clock()	7.463100e-04	2.730845e-02	9.992518e-01	1.205933e+10	

(csv таблица после выполнения программы)



(Гистограмма зависимости времени исполнения от степени оптимизации)



(Гистограмма производительности от степени оптимизации)

Приложение Листинг

main.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#include <string.h>
#include <stdint.h>
#include <math.h>


void getNameCPU(char cpuname[]) {
    FILE *fp;
    if ((fp = fopen("/proc/cpuinfo", "r")) == NULL) {
        printf("Can't open /proc/cpuinfo \n");
        return;
    }
    size_t m = 0;
    char *line = NULL;
    while (getline(&line, &m, fp) > 0) {
        if (strstr(line, "model name")) {
            strcpy(cpuname, &line[13]);
            break;
        }
    }
    for (int i = 0; i < 60; i++)
        if (cpuname[i] == '\n'){
            cpuname[i] = '\0';
            break;
        }
    fclose(fp);
}


int getParameters(int argc, char *argv[], long long &launchCount)
{
    for (int i = 1 ; i < argc ; i++)
    {
        if (strcmp("-l", argv[i]) == 0 || strcmp("--launch-count", argv[i]) == 0)
        {
            i++;
            size_t len = strlen(argv[i]);
            for (size_t j = 0 ; j < len ; j++)
                if (!isdigit(argv[i][j])){
                    printf("Error in arguments: invalid value for --launch-count!\nThe value must be a number!\n");
                    return 1;
                }
            launchCount = atoll(argv[i]);
        }
        else {
            printf("Error in arguments: unknown key \"%s\"\n", argv[i]);
            return 1;
        }
    }
```

```
    return 0;
}
void DGEMM(int** matrixA, int** matrixB, int** matrixC, long long matrixSize){
    for (long long i = 0; i < matrixSize; i++)
        for (long long j = 0; j < matrixSize; j++){
            matrixC[i][j] += 0;
            for (long long k = 0; k < matrixSize; k++)
                matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
        }
}
void DGEMM(long long** matrixA, long long** matrixB, long long** matrixC, long long matrixSize){
    for (long long i = 0; i < matrixSize; i++)
        for (long long j = 0; j < matrixSize; j++){
            matrixC[i][j] += 0;
            for (long long k = 0; k < matrixSize; k++)
                matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
        }
}

void DGEMM(double** matrixA, double** matrixB, double** matrixC, long long matrixSize){
    for (long long i = 0; i < matrixSize; i++)
        for (long long j = 0; j < matrixSize; j++){
            matrixC[i][j] += 0;
            for (long long k = 0; k < matrixSize; k++)
                matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
        }
}

int outToCSV(char* nameCPU, char* type, long long launchCount, double averageTime, double* time, double
dispersion, long long matrixSize)
{
    FILE *fp;
    if (!(fp = fopen("output.csv", "a"))){
        printf("Error: can't open/find output.csv\n");
        return 1;
    }
    fprintf(fp, "%s;%s;%s;%s;%lld;%lld;%s;%e;%e;%e;%e;\n",
            nameCPU,
            "matrixMultiplication",
            type,
            "O0",
            launchCount,
            launchCount * matrixSize * matrixSize,
            "clock()",
            averageTime,
            sqrt(abs(dispersion)),
            abs(dispersion) / averageTime,
            (launchCount * matrixSize * matrixSize) / averageTime);
    fclose(fp);
    return 0;
}

int main(int argc, char *argv[]) {
    srand(time(0));
    long long matrixSize = 100;
    long long launchCount = 100;
```

```
char cpuname[60];
getNameCPU(cpuname);

clock_t start, stop;

if (getParameters(argc, argv, launchCount))
   return 1;
printf("matrixSize = %lld\n", matrixSize);
printf("launchCount = %lld\n", launchCount);

double timeSum = 0;
double dispersion = 0;
double summand1 = 0;
double summand2 = 0;
double time[launchCount];

//////////////////////////
timeSum = 0;
summand1 = 0;
summand2 = 0;


int **matrixA_i = new int*[matrixSize];
int **matrixB_i = new int*[matrixSize];
int **matrixRes_i = new int*[matrixSize];

for (long long i = 0; i < matrixSize; i++) {
   matrixA_i[i] = new int[matrixSize];
   matrixB_i[i] = new int[matrixSize];
   matrixRes_i[i] = new int[matrixSize];
   for (long long j = 0; j < matrixSize; j++) {
      matrixA_i[i][j] = rand() / 10000;
      matrixB_i[i][j] = rand() / 10000;
      matrixRes_i[i][j] = 0;
   }
}

for (long long i = 0; i < launchCount; i++) {
   start = clock();
   DGEMM(matrixA_i, matrixB_i, matrixRes_i, matrixSize);
   stop = clock();
   time[i] = ((double)(stop - start)) / CLOCKS_PER_SEC;
   timeSum += time[i];
   summand1 += time[i] * time[i];
   summand2 += time[i];
}

dispersion = summand1 / launchCount - summand2 / launchCount;

outToCSV(cpuname, (char*)"int**", launchCount, timeSum / launchCount, time, dispersion, matrixSize);

for (long long i = 0; i < matrixSize; i++) {
   delete(matrixA_i[i]);
   delete(matrixB_i[i]);
   delete(matrixRes_i[i]);
}
```

```
    delete[](matrixA_i);
    delete[](matrixB_i);
    delete[](matrixRes_i);
    ////////////////////////
    timeSum = 0;
    summand1 = 0;
    summand2 = 0;

    long long **matrixA_ll = new long long*[matrixSize];
    long long  **matrixB_ll = new long long*[matrixSize];
    long long  **matrixRes_ll = new long long*[matrixSize];

    for (long long i = 0; i < matrixSize; i++) {
        matrixA_ll[i] = new long long[matrixSize];
        matrixB_ll[i] = new long long[matrixSize];
        matrixRes_ll[i] = new long long[matrixSize];
        for (long long j = 0; j < matrixSize; j++) {
            matrixA_ll[i][j] = rand() / 10000;
            matrixB_ll[i][j] = rand() / 10000;
            matrixRes_ll[i][j] = 0;
        }
    }

    for (long long i = 0; i < launchCount; i++) {
        start = clock();
        DGEMM(matrixA_ll, matrixB_ll, matrixRes_ll, matrixSize);
        stop = clock();
        time[i] = ((double)(stop - start)) / CLOCKS_PER_SEC;
        timeSum += time[i];
        summand1 += time[i] * time[i];
        summand2 += time[i];
    }

    dispersion = summand1 / launchCount - summand2 / launchCount;

    outToCSV(cpuname, (char*)"long long**", launchCount, timeSum / launchCount, time, dispersion,
matrixSize);

    for (long long i = 0; i < matrixSize; i++) {
        delete(matrixA_ll[i]);
        delete(matrixB_ll[i]);
        delete(matrixRes_ll[i]);
    }
    delete[](matrixA_ll);
    delete[](matrixB_ll);
    delete[](matrixRes_ll);
    ////////////////////////
    timeSum = 0;
    summand1 = 0;
    summand2 = 0;

    double **matrixA_d = new double*[matrixSize];
    double **matrixB_d = new double*[matrixSize];
    double **matrixRes_d = new double*[matrixSize];

    for (long long i = 0; i < matrixSize; i++) {
```

```
        matrixA_d[i] = new double[matrixSize];
        matrixB_d[i] = new double[matrixSize];
        matrixRes_d[i] = new double[matrixSize];
        for (long long j = 0; j < matrixSize; j++) {
            matrixA_d[i][j] = rand() / 10000 + (double)rand() / RAND_MAX;
            matrixB_d[i][j] = rand() / 10000 + (double)rand() / RAND_MAX;
            matrixRes_d[i][j] = 0;
        }
    }

    for (long long i = 0; i < launchCount; i++) {
        start = clock();
        DGEMM(matrixA_d, matrixB_d, matrixRes_d, matrixSize);
        stop = clock();
        time[i] = ((double)(stop - start)) / CLOCKS_PER_SEC;
        timeSum += time[i];
        summand1 += time[i] * time[i];
        summand2 += time[i];
    }

    dispersion = summand1 / launchCount - summand2 / launchCount;

    outToCSV(cpuname, (char*)"double**", launchCount, timeSum / launchCount, time, dispersion,
matrixSize);

    for (long long i = 0; i < matrixSize; i++) {
        delete(matrixA_d[i]);
        delete(matrixB_d[i]);
        delete(matrixRes_d[i]);
    }
    delete[](matrixA_d);
    delete[](matrixB_d);
    delete[](matrixRes_d);
    return 0;
}
```