Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации Сибирский Государственный Университет

Телекоммуникаций и Информатики СибГУТИ

Кафедра Вычислительных систем

Лабораторная работа №2

По дисциплине “Архитектура вычислительных систем”

Выполнил:

Студент группы ИВ-921 Черемисин И.И.

Работу проверил:

Ассистент кафедры ВС Петухова Я.В.

Новосибирск 2021

# Задание

Реализовать программу для оценки производительности процессора (benchmark).

1.Написать программу(ы)(benchmark) на языке С/С++/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. Можно использовать готовые функции из математической библиотеки (math.h) [3], библиотеки BLAS[4] (англ. Basic Linear Algebra Subprograms —базовые подпрограммы линейной алгебры) и/или библиотеки LAPACK[5](LinearAlgebraPACKage). Обеспечить возможность в качестве аргумента при вызове программы указать общее число испытаний для каждой типовой задачи (минимум 10).

Входные данные для типовой задачи сгенерировать случайным образом.

2.С помощью системного таймера (библиотека time.h, функции clock() илиgettimeofday()) 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 (рассчитать дисперсию и среднеквадратическое отклонение).

3.Результаты испытаний в самой программе (или с помощью скрипта) сохранить в файл в формате CSV со следующей структурой:

[PModel; Task;OpType;Opt;InsCount;Timer;Time;LNum;AvTime;AbsErr;RelE rr;TaskPerf], где

Pmodel – ProcessorModel, модель процессора,на котором проводятся испытания;

Task–название выбранной типовой задачи (например, sin, log, saxpy, dgemv, sgemmи др.);

OpType– OperandType, тип операндов, используемых при вычислениях типовой задачи;

Opt–Optimisations, используемы ключи оптимизации (None, O1, O2 и др.);

InsCount–InstructionCount, оценка числа инструкций при выполнении типовой задачи;

Timer–название функции обращения к таймеру (для измерения времени);

Time–время выполнения отдельного испытания;

LNum–LaunchNumer, номер испытаниятиповой задачи.

AvTime–AverageTime, среднее время выполнения типовой задачи из всех испытаний[секунды];

AbsError– AbsoluteError, абсолютная погрешность измерения времени в секундах;

RelError–RelativeError, относительная погрешность измерения времени в %;

TaskPerf–TaskPerformance, производительность (быстродействие) процессора при выполнении типовой задачи.

3. \* Оценить среднее время испытания каждой типовой задачи с разным типом входных данных (целочисленные, с одинарной и двойной точностью).

4.Построить сводную диаграмму производительности в зависимости от задач и выбранных исходных параметров испытаний.Оценить среднее быстродействие

(производительность)для равновероятного использования типовых задач.

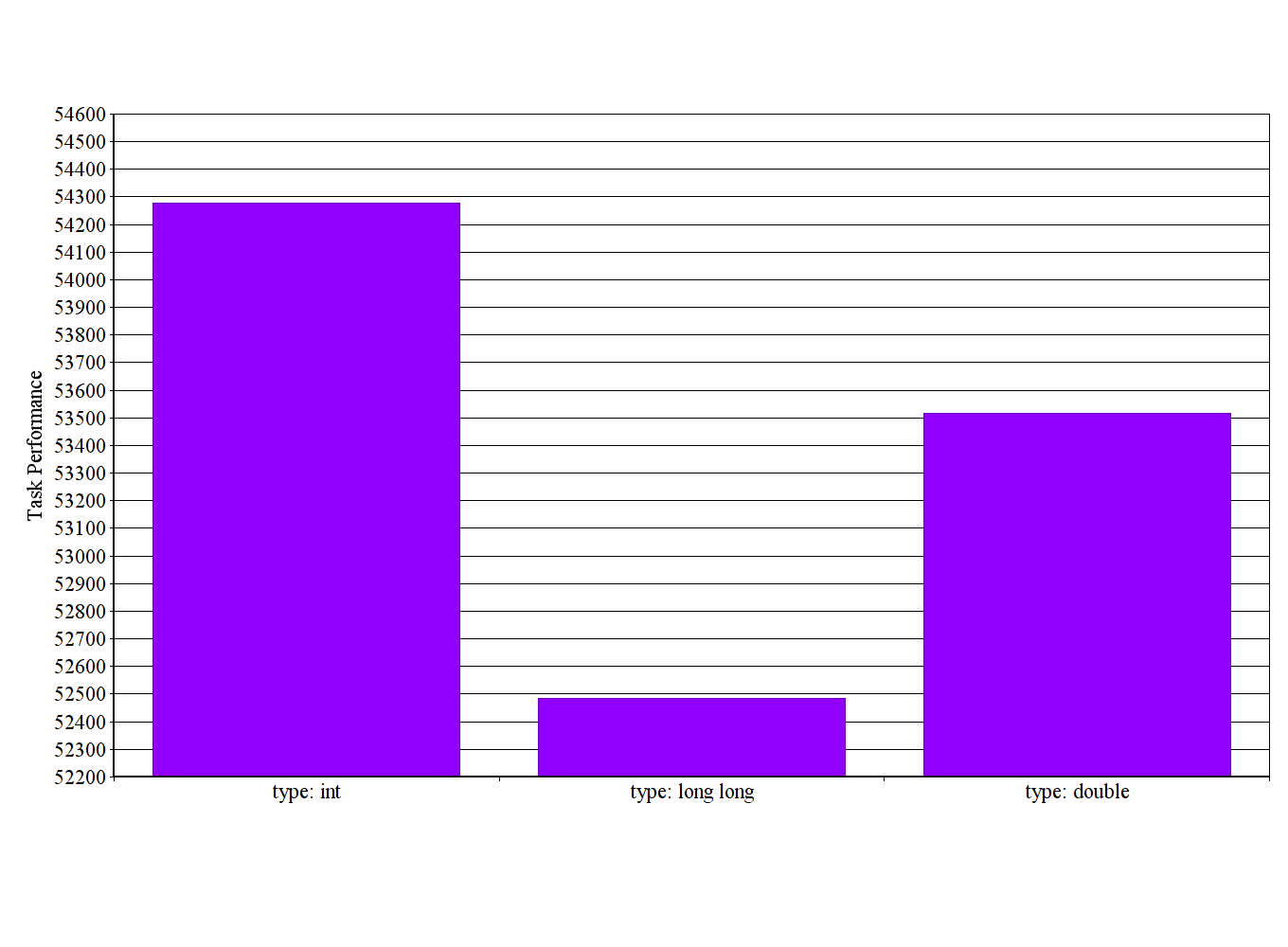
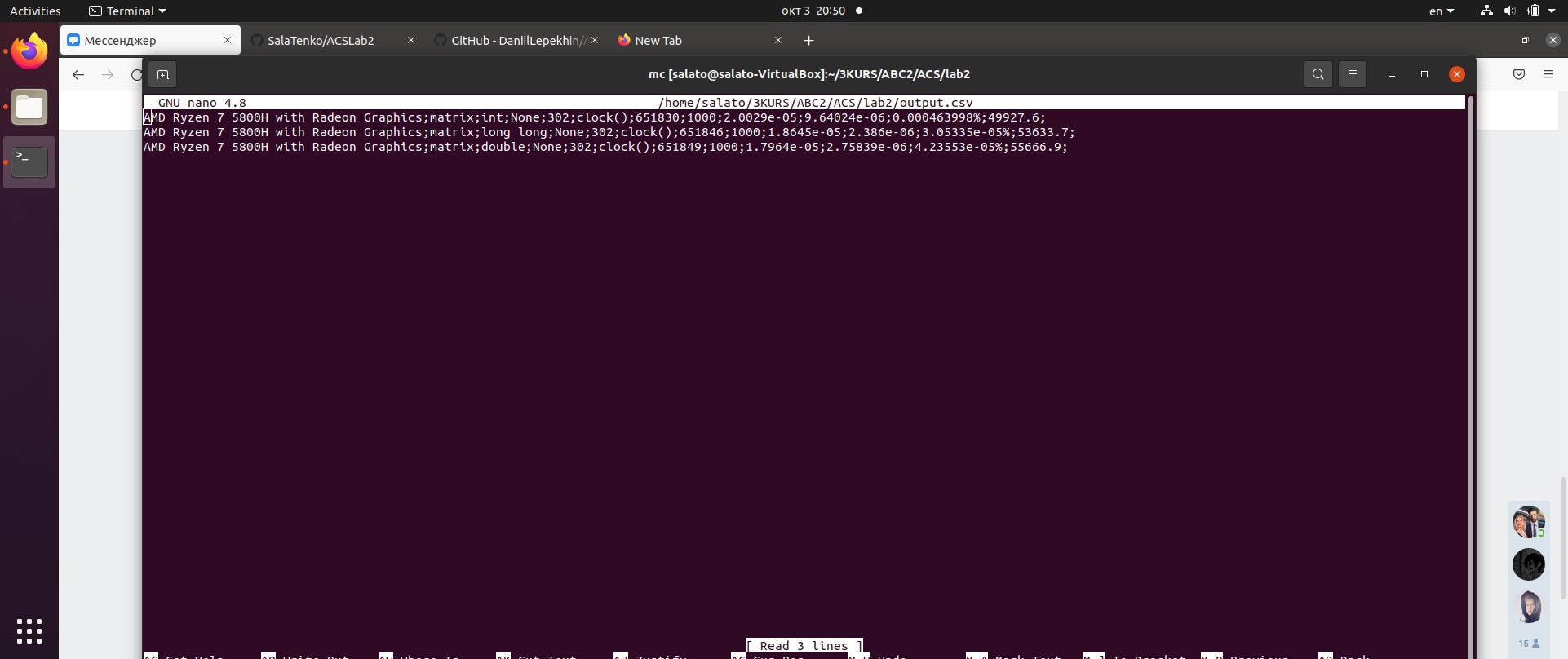
# Ход работы

Я взял функцию по работе с матрицами и векторами данных с несколькими типами данных для оценки производительности из библиотеки math.h. Чтобы найти эталонное время каждой функции, я вызвал каждую функцию тысячу раз и записал каждый результат в массив. Далее я реализовал функции test1\_1/2/3, в которых в свою очередь реализовал типовые задачи с разными типами данных int/long long/double, в которых я создаю матрицу и заполняю ее случайными числами с помощью функции rand (), затем с помощью функции clock () определяю время работы каждого теста и записываем эталонное время.

Также программа находит эталонное время для каждого типа данных (int, long long, double) и для ключа оптимизации при компиляции (None) перед вычислением исследуемого времени. Таким образом абсолютная и относительная погрешность становится более точной. Ключ None вместе с большим количеством вызовов функций (хотя бы 1000) позволяют хоть как-то найти хорошее отношение эталонного времени и исследуемого времени, и найти адекватную погрешность.

В конце работы программы, вся нужная информация выводиться в файл output.csv.

**Итог работы**



**Листинг**

Foo.c

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119 | #include "foo.h"  **void** **test1\_1**(**int** mx, **double** &time) {  **int** arr[mx][mx];  **int** i, j;  **for** (i = **0**; i < mx; i++) {  **for** (j = **0**; j < mx; j++) {  arr[i][j] = rand() % mx;  }  }  **int** koef = rand() % mx;  **clock\_t** start, stop;  start = clock();  **for** (i = **0**; i < mx; i++) {  **for** (j = **0**; j < mx; j++) {  arr[i][j] \*= koef;  }  }  stop = clock();  time = ((**double**)(stop - start)) / CLOCKS\_PER\_SEC;  }  **void** **test1\_2**(**int** mx, **double** &time) {  **long** **long** arr[mx][mx];  **int** i, j;  **for** (i = **0**; i < mx; i++) {  **for** (j = **0**; j < mx; j++) {  arr[i][j] = rand() % mx;  }  }  **long** **long** koef = rand() % mx;  **clock\_t** start, stop;  start = clock();  **for** (i = **0**; i < mx; i++) {  **for** (j = **0**; j < mx; j++) {  arr[i][j] \*= koef;  }  }  stop = clock();  time = ((**double**)(stop - start)) / CLOCKS\_PER\_SEC;  }  **void** **test1\_3**(**int** mx, **double** &time) {  **double** arr[mx][mx];  **int** i, j;  **for** (i = **0**; i < mx; i++) {  **for** (j = **0**; j < mx; j++) {  arr[i][j] = rand() % mx \* (rand() % **10** \* **0.1**);  }  }  **double** koef = rand() % mx \* (rand() % **10** \* **0.1**);  **clock\_t** start, stop;  start = clock();  **for** (i = **0**; i < mx; i++) {  **for** (j = **0**; j < mx; j++) {  arr[i][j] \*= koef;  }  }  stop = clock();  time = ((**double**)(stop - start)) / CLOCKS\_PER\_SEC;  }  **void** **benchmark**(**int** num, **long** **long** n, **int** matrix\_mx,  **double** &avg\_time, **double** &dispersion,  **double** &abs\_error, **double** &rel\_error)  {  **double** summand1 = **0**, summand2 = **0**;  **double** x;  **for** (**long** **long** i = **0**; i < n; i++) {  **switch** (num) {  **case** **0**:  test1\_1(matrix\_mx, x);  **break**;  **case** **1**:  test1\_2(matrix\_mx, x);  **break**;  **case** **2**:  test1\_3(matrix\_mx, x);  **break**;  **default:**  printf("ERROR: wrong **\"**num**\"** in benchmark() **\n**");  **break**;  }  summand1 += x \* x;  summand2 += x;  }  summand1 /= n;  summand2 /= n;  avg\_time = summand2;  summand2 \*= summand2;  dispersion = summand1 - summand2; abs\_error = sqrt(dispersion);  rel\_error = dispersion / avg\_time;  }  **void** **get\_cpu\_name**(**char** cpuname[]) {  **FILE** \*fcpu;  **if** ((fcpu = fopen("/proc/cpuinfo", "r")) == NULL) {  printf("Can't open /proc/cpuinfo **\n**");  **return**;  }  **size\_t** m = **0**;  **char** \*line = NULL;  **while** (getline(&line, &m, fcpu) > **0**) {  **if** (strstr(line, "model name")) {  strcpy(cpuname, &line[**13**]);  **break**;  }  }  **for** (**int** i = **0**; i < **50**; i++) {  **if** (cpuname[i] == '\n')  cpuname[i] = '\0';  }  fclose(fcpu);  } |

Foo.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #ifndef FOO  #define FOO  #include <math.h>  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <string.h>  **void** **benchmark**(**int** num, **long** **long** n, **int** matrix\_mx,  **double** &avg\_time, **double** &dispersion,  **double** &abs\_error, **double** &rel\_error);  **void** **get\_cpu\_name**(**char** cpuname[]);  #endif |

Main.cpp

#include "foo.h"

**int** **main**() {

srand(time(**0**));

**int** cnt\_tests = **3**, i;

**long** **long** n[cnt\_tests];

n[**0**] = n[**1**] = n[**2**] = **1000**;

**double** avg\_time[cnt\_tests], dispersion[cnt\_tests];

**double** abs\_error[cnt\_tests], rel\_error[cnt\_tests];

**int** matrix\_mx = **100**;

**for** (i = **0**; i < cnt\_tests; i++) {

benchmark(i, n[i], matrix\_mx, avg\_time[i], dispersion[i], abs\_error[i], rel\_error[i]);

}

**char** cpuname[**50**] = {'\0'};

get\_cpu\_name(cpuname);

**FILE** \*fout;

**if** ((fout = fopen("output.csv", "w")) == NULL) {

printf("Can't open output.csv **\n**");

**return** **1**;

}

**for** (i = **0**; i < cnt\_tests; i++) {

fprintf(fout, cpuname);

fprintf(fout, ";");

**switch** (i) {

**case** **0**:

fprintf(fout, "matrix;int;None;");

**break**;

**case** **1**:

fprintf(fout, "matrix;long long;None;");

**break**;

**case** **2**:

fprintf(fout, "matrix;double;None;");

**break**;

**default:**

printf("ERROR: wrong **\"**num**\"** while writing a file **\n**");

**break**;

}

**switch** (i) {

**case** **0**:

fprintf(fout, "%d;", matrix\_mx \* **3** + **2**);

**break**;

**case** **1**:

fprintf(fout, "%d;", matrix\_mx \* **3** + **2**);

**break**;

**case** **2**:

fprintf(fout, "%d;", matrix\_mx \* **3** + **2**);

**break**;

}

fprintf(fout, "clock();");

fprintf(fout, "%ld;", clock());

fprintf(fout, "%lld;", n[i]);

fprintf(fout, "%g;", avg\_time[i]);

fprintf(fout, "%g;", abs\_error[i]);

fprintf(fout, "%g%%;", rel\_error[i] \* **100**);

fprintf(fout, "%g;", **1** / avg\_time[i]);

fprintf(fout, "**\n**");

}

fclose(fout);

**return** **0**;

}

Makefile

COMP = g++

FLAGS = -Wall -o

OBJS = main.cpp foo.cpp

.PHONY: clean run all

**all:** main

**main:**

$(COMP) $(OBJS) $(FLAGS) main -lm