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
Орлова А.А.

Работу проверил:
Ассистент кафедры ВС
Петухова Я.В.

Новосибирск 2021

Оглавление

Постановка задачи

Реализовать 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. Можно использовать готовые функции из математической библиотеки (math.h) [3], библиотеки BLAS [4] (англ. Basic Linear Algebra Курс «Архитектура вычислительных систем». СибГУТИ. 2020 г. Subprograms — базовые подпрограммы линейной алгебры) и/или библиотеки LAPACK [5] (Linear Algebra PACKage). 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;RelErr;TaskPerf], где
   PModel – Processor Model, модель процессора, на котором проводятся испытания;
   Task – название выбранной типовой задачи (например, sin, log, saxpy, dgemv, sgemm и др.);
   OpType – Operand Type, тип операндов используемых при вычислениях типовой задачи;
   Opt – Optimisations, используемы ключи оптимизации (None, O1, O2 и др.);

InsCount – Instruction Count, оценка числа инструкций при выполнении типовой задачи;
Timer – название функции обращения к таймеру (для измерения времени);
Time – время выполнения отдельного испытания;
LNum – Launch Numer, номер испытания типовой задачи.
AvTime –Average Time, среднее время выполнения типовой задачи из всех испытаний[секунды];
AbsError – Absolute Error, абсолютная погрешность измерения времени в секундах;
RelError – Relative Error, относительная погрешность измерения времени в %;
TaskPerf – Task Performance, производительность (быстродействие) процессора при выполнении типовой задачи.

3.1. * Оценить среднее время испытания каждой типовой задачи с разным типом входных данных (целочисленные, с одинарной и двойной точностью).
3.2. ** Оценить среднее время испытания каждой типовой задачи с оптимизирующими преобразования исходного кода компилятором (ключи –O1, O2, O3 и др.).
3.3. *** Оценить и постараться минимизировать накладные расходы(время на вызов функций, влияние загрузки системы и т.п.) при испытании, то есть добиться максимальной точности измерений.

4. Построить сводную диаграмму производительности в зависимости от задач и выбранных исходных параметров испытаний. Оценить среднее быстродействие (производительность) для равновероятного использования типовых задач.

Выполнение Работы

ЯиспользовалаязыкC.Bmain()задаетсяколичество видов тестов (test_size, равное трём), количество испытаний типовых задач 10 исоздаютсямассивыдляхраненияреднеговременивыполнениятиповойзадачи,точностьоценкивремени,погрешности,общеевремявыполнениятеста.Мои типовые задачи — это умножение двух матриц. Также задаетсяфиксированныйразмерматрицы.

Далее в цикле вызывается каждый тест (тесты отличаются друг от другатипом данных, создаваемых матриц). Тесты выполняются черезфункциюbenchmark().Послеэтогооткрываетсяфайloutput.csv,вкоторыйзаписываютсяданныевсоответствиисзаданием.

Вфункцииbenchmark()вызываетсяmatrix_int(),matrix_double()илиmatrix_float() взависимости от входных данных. Последние три функции работают последующему алгоритму: создаются два массива, которые заполняются рандомнымичислами и третий массив(куда будут записываться результаты умножения двух матриц), который заполняется нулями, после чегозамеряется времяивыполняетсяумножение двух матриц. По окончанию этого снова замеряется время stop =clock().Вычисляетсявремявсекундах.

Выполнив нужную типовую задачу, измеренное время добавляется кдвум переменным (summand1 и summand2), которые нужны для измерениядисперсии. Дисперсия вычисляетсяподаннойформуле:

$$D(X) = M\left(X^2\right) - M^2\left(X\right).$$

гдеM(X)—этоматожидание.

Далее находится среднее квадратическое отклонение путем извлечениякорня из дисперсии. Дисперсия — это точность измерения, среднееквадратическое отклонение — абсолютная погрешность. Поделив дисперсиюнасреднее время выполнениятиповойзадачиполучимотносительнуюпогрешность.

Ход работы со *

*

Какбылоописановыше,типоваязадачаимееттритипа,аименно:int,double, float. Скорость выполнения зависит от заданного типа. Это особеннозаметно при использовании оптимизации. Увидеть это можно на скриншотах вразделе «Результатработы».

**

Еслизапускатьпрограммубезключакомпиляциитовремявыполнениятиповых задач в зависимости от типа данных может отличаться. Если запускатьпрограмму с любым ключом оптимизации (O1, O2, O3), то хорошо заметно, чтосреднеевремявыполнениятиповойзадачиуменьшается.
При ключе O1 компилятор попытается сгенерировать быстрый, занимающийменьшеобъемакод,беззатрачиваниянаибольшеговременикомпиляции.
O2активируетнесколькодополнительныхфлаговвдобавоккфлагам,активированныхO1.СпараметромO2,компиляторпопытаетсяувеличитьпроизводительность кода без нарушения размера, и без затрачивания большогоколичества времени
компиляции.O3Включаетоптимизации,являющейсядорогостоящейсточкизрениявремени компиляции и потребления памяти, что порой приводит к замедлениюсистемы из-за больших двоичных файлов и увеличения потребления памяти.

***

Я постаралась минимизировать накладные расходы на измерение времени длятиповыхзадачтем,чтовремязамеряетсянепосредственнопередипослеумножениядвухматриц.

Результат работы

1	Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz	matrixA * matrixB	int	None	clock()	10	0.0053993	0.0015287	0.043282%	185.209
2	Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz	matrixA * matrixB	double	None	clock()	10	0.0045631	2.37926e-05	1.24058e-05%	219.149
3	Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz	matrixA * matrixB	float	None	clock()	10	0.0046988	7.18053e-06	1.0973e-06%	212.82

Рисунок1:файлoutput.csvпослевыполненияпрограммы

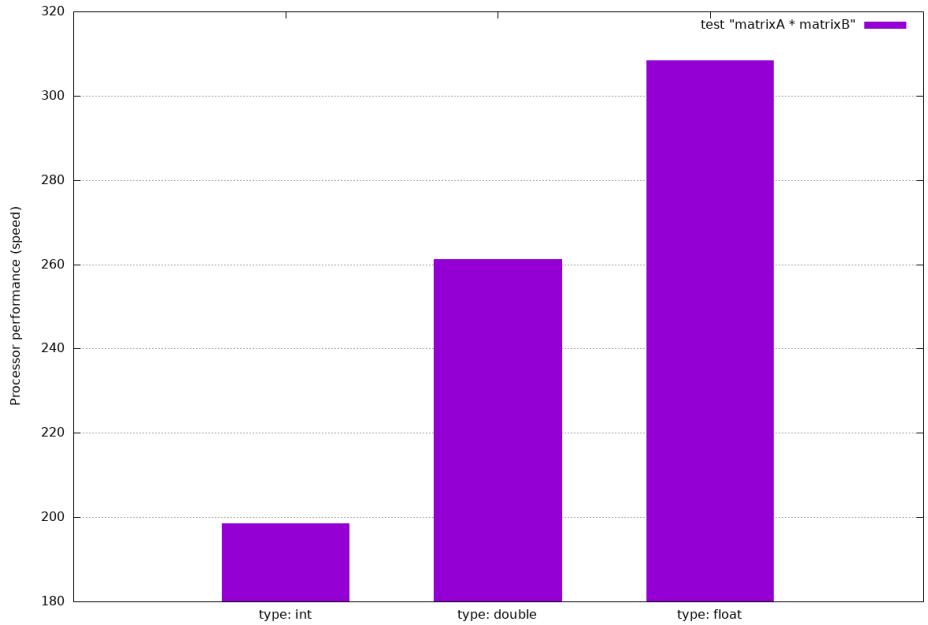

Рисунок2:диаграммапроизводительностибезключейоптимазации

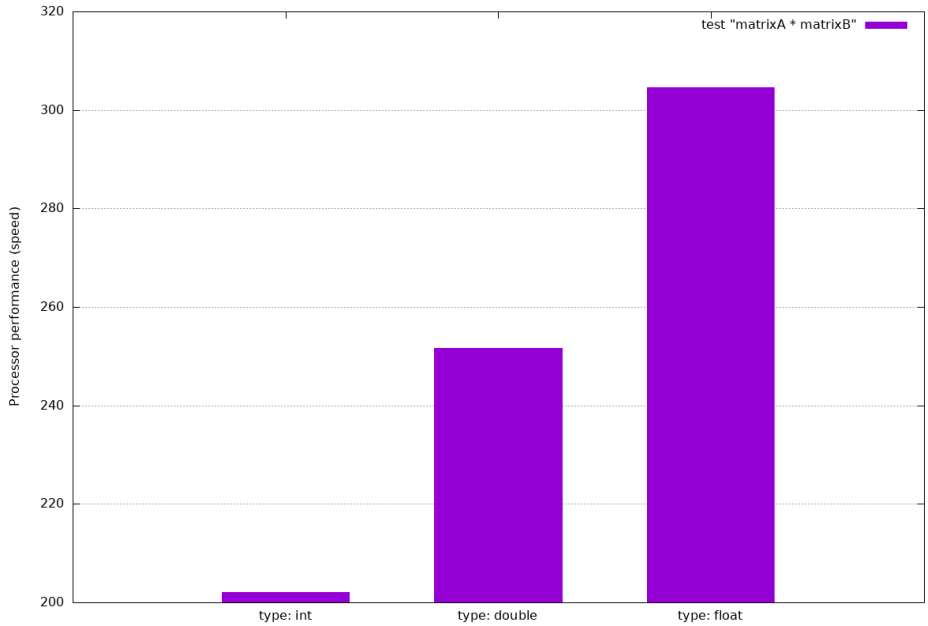

Рисунок3:диаграммапроизводительностисключом-O1

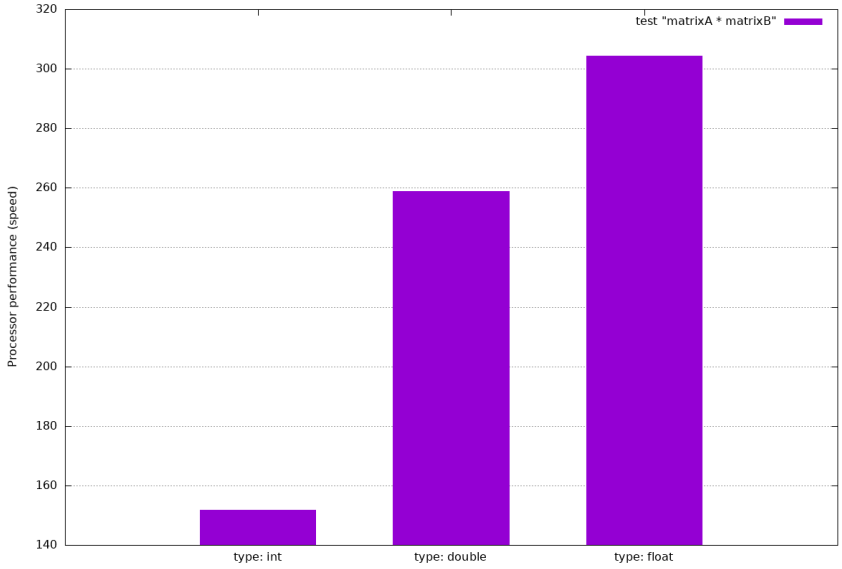

Рисунок4:диаграммапроизводительностисключом-O2

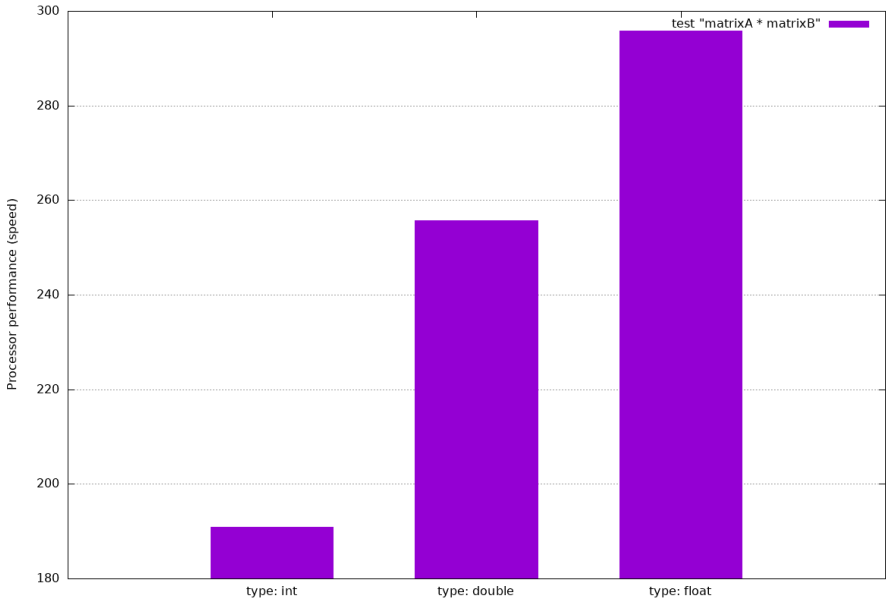

Рисунок5:диаграммапроизводительностисключом-O3

Листинг

Файлmain.cpp

```
#include"foo.h"
intmain() {
srand(time(0));
int num_test = 3;
int n = 10;
double avg_time[num_test], dispersion[num_test];
double abs_error[num_test], rel_error[num_test];
int matrix_size = 100;
for (int i = 0; i < num_test; i++) {
benchmark(i, n, matrix_size, 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");
return1;
}
for (int i = 0; i < num_test; i++) {
fprintf(fout, cpuname);
fprintf(fout, ";");
switch (i) {
case0:
fprintf(fout, "matrixA * matrixB;int;None;");
break;
case1:
fprintf(fout, "matrixA * matrixB;double;None;");
break;
case2:
fprintf(fout, "matrixA * matrixB;float;None;");
break;
default:
printf("Error writing to file.\n");
break;
}
fprintf(fout, "clock();");
fprintf(fout, "%d;", n);
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);
return0;
}
```

Файлfoo.h

```
#ifndef FOO
#define FOO

#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

voidbenchmark(int num_test, int n, int matrix_size, double&avg_time,
double&dispersion, double&abs_error, double&rel_error);
voidget_cpu_name(char cpuname[]);

#endif
```

Файлfoo.cpp

```
#include “foo.h”
voidtest1_1(int n, double&time) {
int a[n][n], b[n][n], c[n][n];
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
a[i][j] = rand() % 10;
b[i][j] = rand() % 10;
}
}
clock_t start, stop;
start = clock();
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
c[i][j] = 0;
for (int k = 0; k < n; k++) {
c[i][j] += a[i][k] * b[k][j];
}
}
}
stop = clock();
time = ((double)(stop - start)) / CLOCKS_PER_SEC;
}
voidtest1_2(int n, double&time) {
double a[n][n], b[n][n], c[n][n];
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
a[i][j] = rand() % 10;
b[i][j] = rand() % 10;
}
}
clock_t start, stop;
start = clock();
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
c[i][j] = 0;
for (int k = 0; k < n; k++) {
c[i][j] += a[i][k] * b[k][j];
```

```
}
}
}
stop = clock();
time = ((double)(stop - start)) / CLOCKS_PER_SEC;
}

voidtest1_3(int n, double&time) {
float a[n][n], b[n][n], c[n][n];


for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
a[i][j] = rand() % 10;
b[i][j] = rand() % 10;
}
}
clock_t start, stop;
start = clock();
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
c[i][j] = 0;
for (int k = 0; k < n; k++) {
c[i][j] += a[i][k] * b[k][j];
}
}
}
stop = clock();
time = ((double)(stop - start)) / CLOCKS_PER_SEC;
}
// Производительность процессора
voidbenchmark(int num_test, int n, int matrix_size,
double&avg_time, double&dispersion,
double&abs_error, double&rel_error)
{
double summand1 = 0, summand2 = 0;
double x;
for (int i = 0; i < n; i++) {
switch (num_test) {
case0:
test1_1(matrix_size, x);
break;
case1:
test1_2(matrix_size, x);
break;
case2:
test1_3(matrix_size, x);
break;
default:
printf("ERROR: wrong \"num_test\" 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;
}

voidget_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);
}
```

Файлdiagram.gpi

```
#! /usr/bin/gnuplot
#! /usr/bin/gnuplot -persist
set terminal png font "Verdana,12" size 1200, 800
set output "diagram_O3.png"
set datafile separator ';'
set ylabel "Processor performance (speed)"
set grid ytics
set xrange [-1:3]
set xtics ("type: int" 0, "type: double" 1, "type: float" 2,)
set style data boxes
set boxwidth 0.6 absolute
set style fill solid 1
plot"output.csv"using10title"test \"matrixA * matrixB\""
```