

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

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() или gettimeofday()) 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;Abs Err;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, используемые 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.

Результат работы

В ходе данной лабораторной работы было реализовано три теста, основанных на умножение двух матриц. Тесты отличаются только типом данных, используемых для создания матриц. Результат выполнения программы записывается в файл output.csv в виде таблицы.

PModel	task	OpType	Opt	Timer	LNum	AVTime	ABSErr	RELErr	TaskPerf
Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz	matrixA * matrixB	OpType: int	Opt: None	clock()	10	0.00472	0.00109311	0.0253153%	211.864
Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz	matrixA * matrixB	OpType: double	Opt: None	clock()	10	0.004035	0.000299462	0.00222249%	247.831
Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz	matrixA * matrixB	OpType: float	Opt: None	clock()	10	0.0035751	0.000267111	0.00199569%	279.712

Таблица 1. Файл output.csv

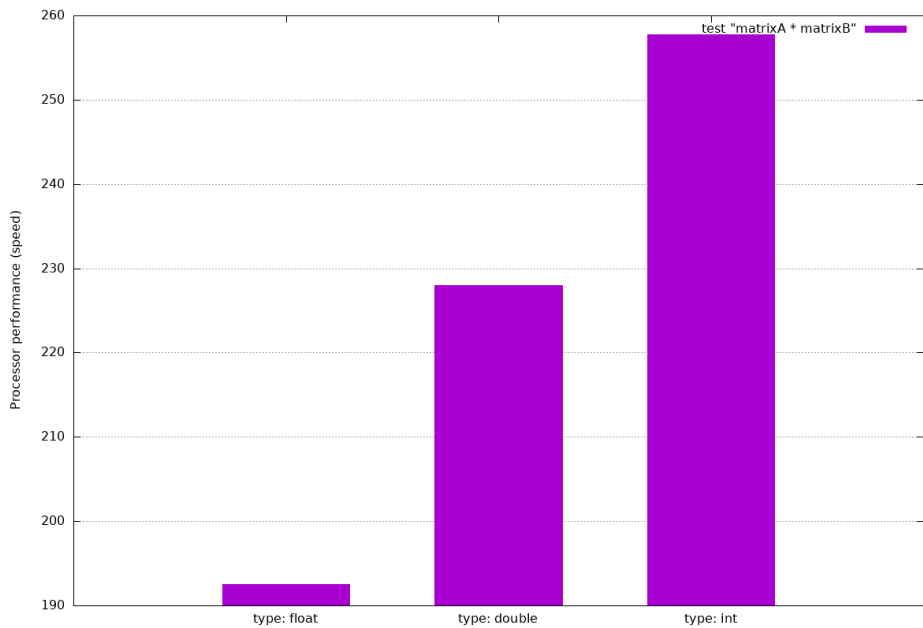


График 1. Производительность процессора без ключей оптимизации

При ключе O1 компилятор попытается сгенерировать быстрый, занимающий меньше объема код, без затрачивания наибольшего времени компиляции.

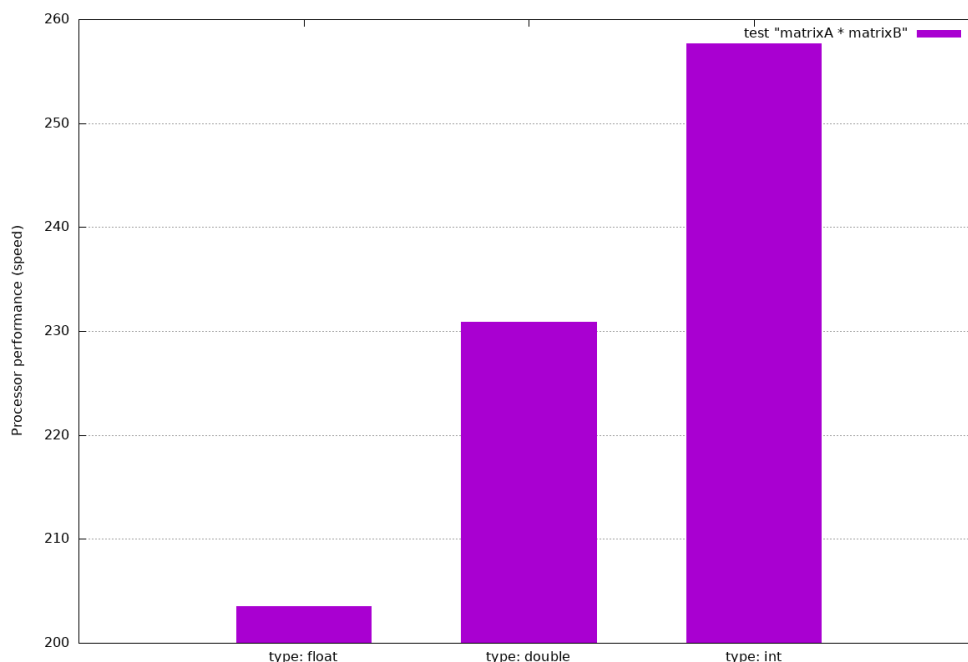


График 2. Производительность процессора с ключом -O1

O2 активирует несколько дополнительных флагов вдобавок к флагам, активированным O1. С параметром O2, компилятор попытается увеличить производительность кода без нарушения размера, и без затрат времени компиляции.

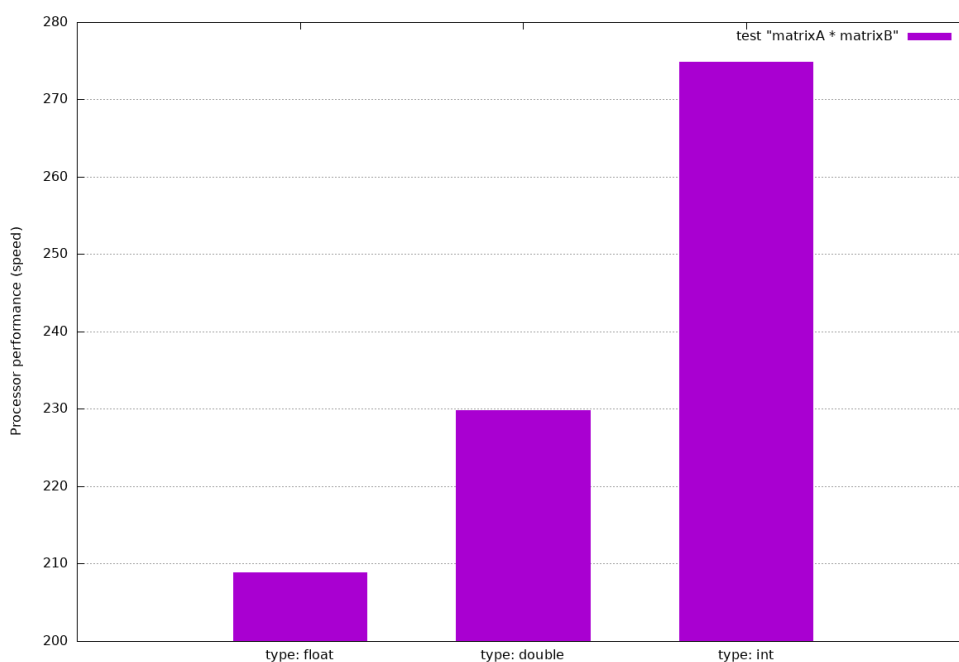


График 2. Производительность процессора с ключом -O2

ОЗ включает оптимизации, являющейся дорогостоящей с точки зрения времени компиляции и потребления памяти, что приводит к замедлению системы из-за больших двоичных файлов и увеличения потребления памяти.

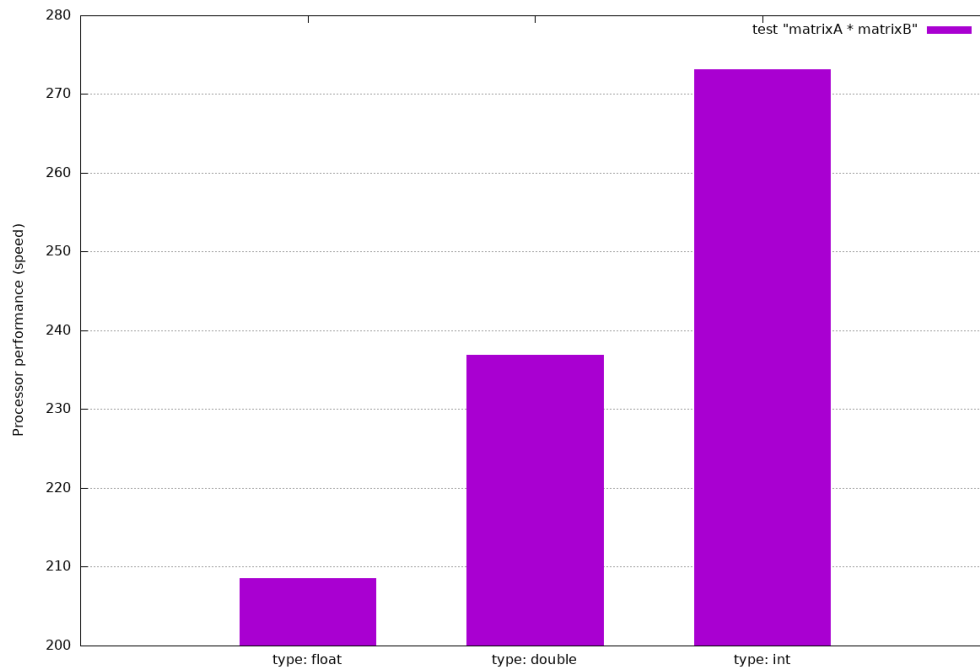


График 3. Производительность процессора с ключом -O3

ЛИСТИНГ

Файл main.cpp

```
#include "foo.h"
int main() {
    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");
        return 1;
    }
    for (int i = 0; i < num_test; i++) {
        fprintf(fout, cpuname);
        fprintf(fout, ";");
        switch (i) {
            case 0:
                fprintf(fout, "matrixA * matrixB;int;None;");
                break;
            case 1:
                fprintf(fout, "matrixA * matrixB;double;None;");
                break;
            case 2:
                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);
    return 0;
}
```

Файл foo.h

```

#ifndef F00
#define F00

#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

```

Файл foo.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;
}

void test1_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
void benchmark(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) {
case 0:
test1_1(matrix_size, x);
break;
case 1:
test1_2(matrix_size, x);
break;
case 2:
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;
}

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);
}

```

Файл diagram.gpi

```

#!/usr/bin/gnuplot
#!/usr/bin/gnuplot -persist
set terminal png font "Verdana,12" size 1200, 800
set output "diagram_03.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" using 1:0 title "test \"matrixA * matrixB\""

```