ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

«Оценка характеристик персонального компьютера (ПК)»

по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. ИП-81
Разумов Д.Б.
Проверил: ст. преп. Кафедры ВС
Ткачёва Т.А.

Новосибирск 2020

Содержание

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 библиотек. Обеспечить возможность подать на вход программы 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;LNum;InsCount;Timer;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.);

LNum – Launch Numer, число испытаний типовой задачи;

InsCount – Instruction Count, оценка числа инструкций при выполнении типовой задачи;

Timer – название функции измерения времени;

AvTime – Average Time, среднее время выполнения типовой задачи в секундах;

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

3. ** Оценить среднее время испытания каждой типовой задачи с оптимизирующими преобразования исходного кода компилятором (ключи –O1, O2, O3 и др.).

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

Я преимущество использовал язык C. В начале main() задается количество видов тестов (cnt_tests), количество испытаний типовых задач (n) и создаются массивы для хранения среднего времени выполнения типовой задачи, точность оценки времени, погрешности, общее время выполнения теста. Мои типовые задачи — это умножение матрицы на число. Также задается фиксированный размер матрицы.

Далее в цикле вызывается каждый тест (тесты отличаются друг друга типом данных, на которых создается матрица). Тесты выполняются через функцию benchmark(), описание которой будет ниже. После этого открывается файл output.csv, в который записываются данные в соответствии с заданием.

В функции benchmark() вызывается test1_1(), test1_2() или test1_3() в зависимости от входных данных. Последние три функции работают по следующему алгоритму: создается массив, который заполняется рандомными числами, затем создается рандомное число (koef), после чего замеряется время (start = clock()) и выполняется умножение каждого элемента массива на число (это и есть типовая задача). По окончанию этого снова замеряется время (stop = clock()). Вычисляется время в секундах, которое сохраняется в одном из параметров (double &time) функции.

test1_1(), test1_2() или test1_3() отличаются друг друга только типом данных, используемых для создания матрицы и числа, на которое она умножается.

Выполнил нужную типовую задачу, измеренное время добавляется к двум переменным (summand1 и summand2), которые нужны для измерения дисперсии. Дисперсия вычисляется по данной формуле:

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

где M(X) — это матожидание.

Далее находится среднее квадратическое отклонение путем извлечения корня из дисперсии. Дисперсия — это точность измерения, среднее квадратическое отклонение — абсолютная погрешность. Поделив дисперсию на среднее выполнение типовой задачи получим относительную погрешность.

Диаграмма производительности выводится с помощью скрипта на gnuplot.

Выполнение условий с *

*

Как было описано выше, типовая задача имеет три типа, а именно: int, long long, double. Скорость выполнения зависит от заданного типа. Это особенно заметно при использовании оптимизации. Увидеть это можно на скриншотах в разделе «Результат работы».

**

Если запускать программу без ключа компиляции то время выполнения типовых задач в зависимости от типа данных может отличаться. Если запускать программу с любым ключом оптимазации (O1, O2, O3), то хорошо заметно, что среднее время выполнения типовой задачи уменьшается.

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

O2 активирует несколько дополнительных флагов вдобавок к флагам, активированных O1. С параметром O2, компилятор попытается увеличить производительность кода без нарушения размера, и без затрачивания большого количества времени компиляции. На этом уровне могут быть использованы SSE и AVX, но YMM-регистры не будут использоваться.

O3 Включает оптимизации, являющейся дорогостоящей с точки зрения времени компиляции и потребления памяти, что порой приводит к замедлению системы из-за больших двоичных файлов и увеличения потребления памяти. циклы в коде векторизируются и используют регистры AVX YMM.

***

Я постарался минимизировать накладные расходы на измерение времени для типовых задач тем, что время замеряется непосредственно перед и после умножением числа на матрицу.

Результат работы

Рисунок 1: файл output.csv после выполнения программы

	A	B	C	D	E	F	G	H	I	J	K
1	Intel(R) Core(TM) i3-8130U	num * matrix	int	None	1000	302	clock()	3.9965e-05	6.13692e-06	9.42369e-05%	25021.9
2	Intel(R) Core(TM) i3-8130U	num * matrix	long long	None	1000	302	clock()	2.6718e-05	2.56095e-06	2.4547e-05%	37428
3	Intel(R) Core(TM) i3-8130U	num * matrix	double	None	1000	302	clock()	2.4982e-05	2.58373e-06	2.67219e-05%	40028.8

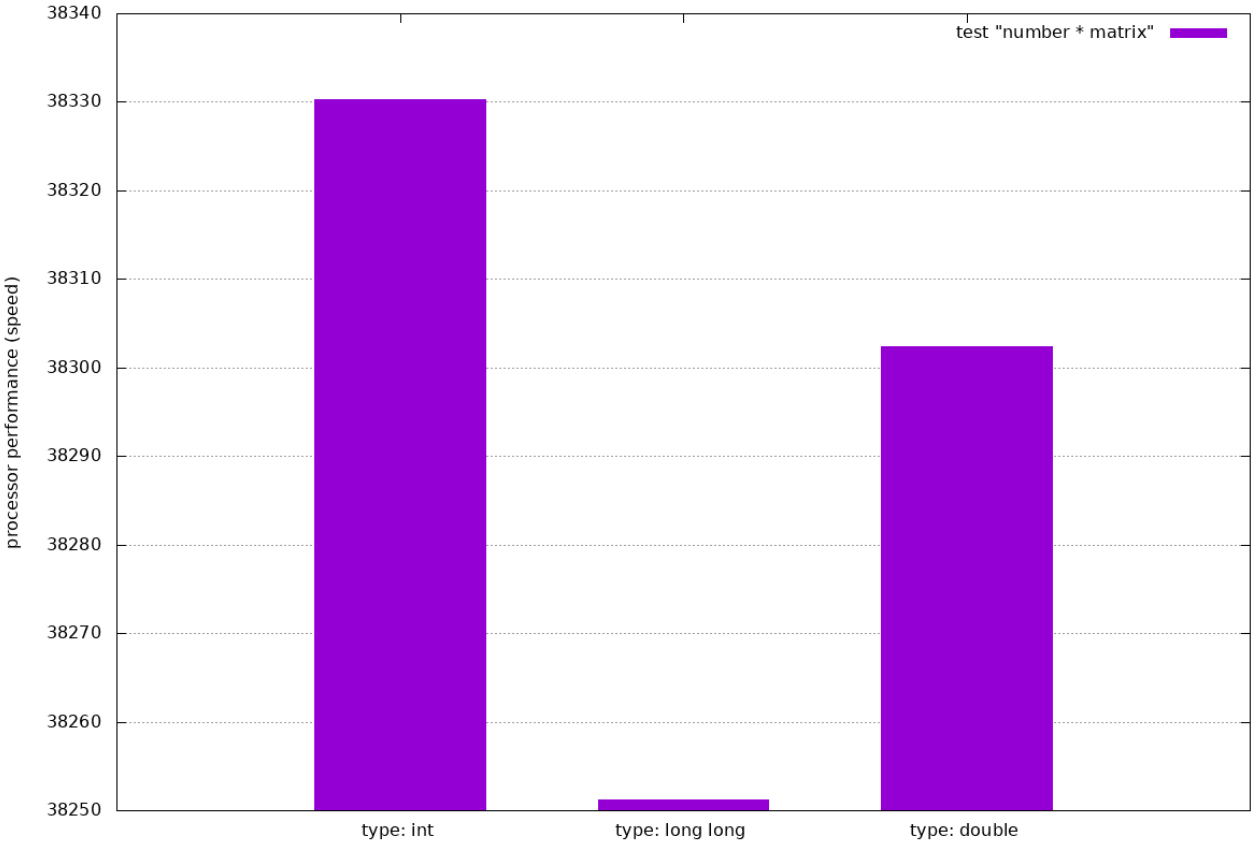

Рисунок 2: диаграмма производительности без ключей оптимазации

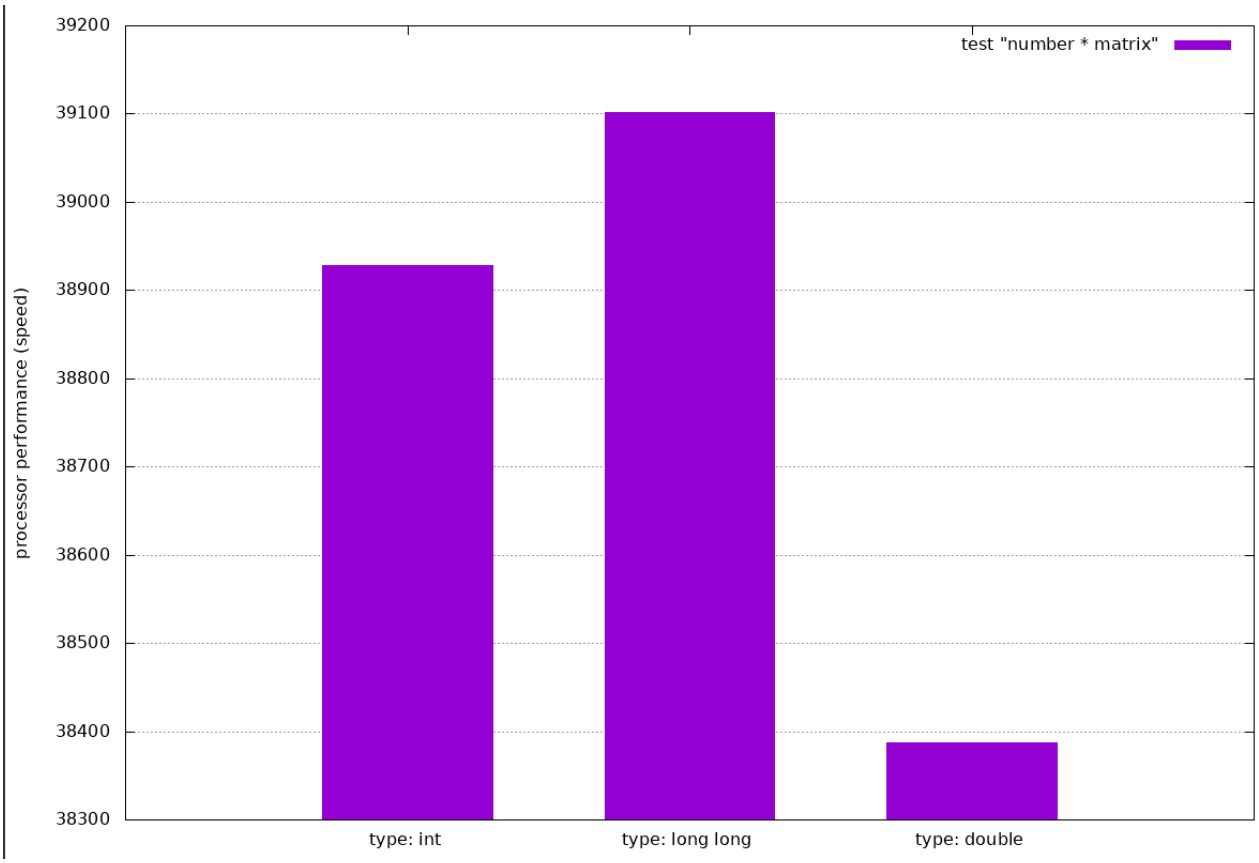

Рисунок 3: диаграмма производительности без ключей оптимазации (перезапуск программы и скрипта)

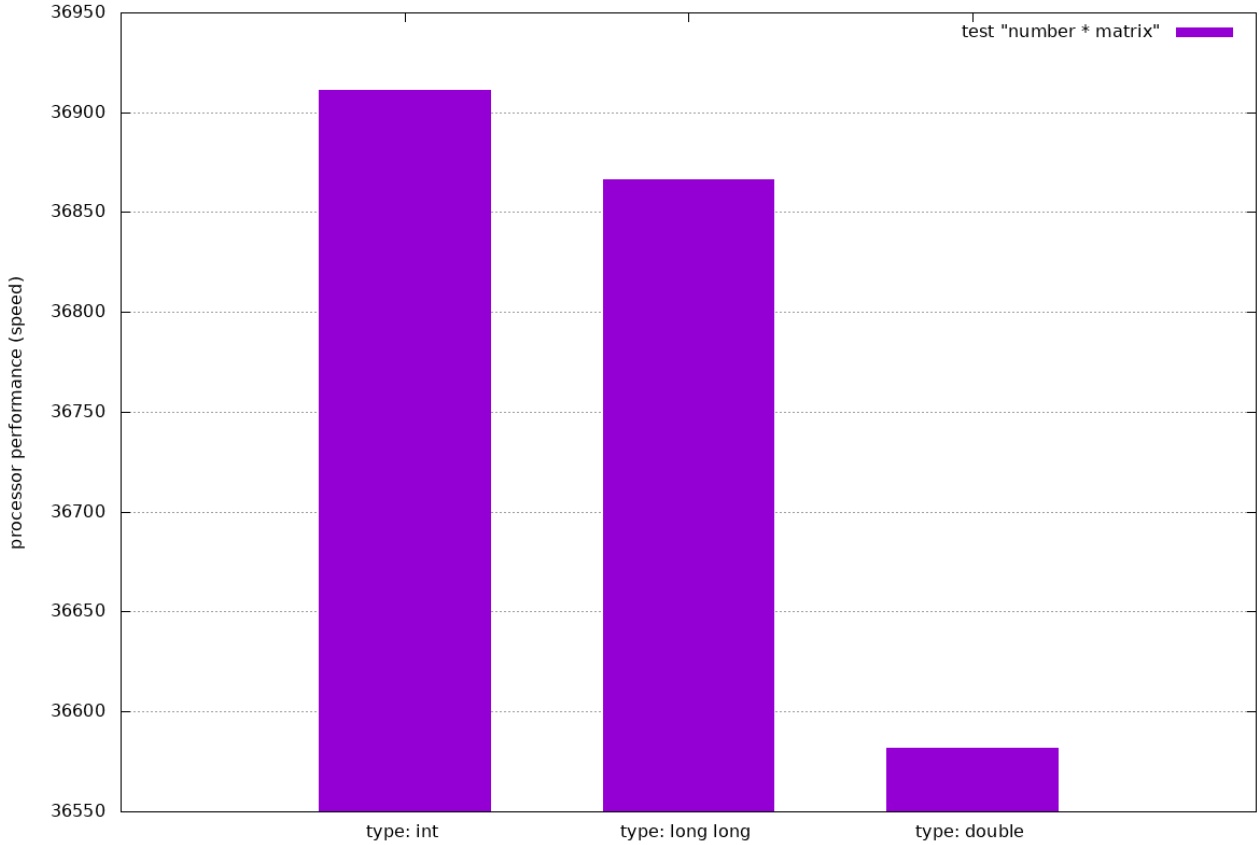

Рисунок 4: диаграмма производительности без ключей оптимазации (еще один перезапуск программы и скрипта)

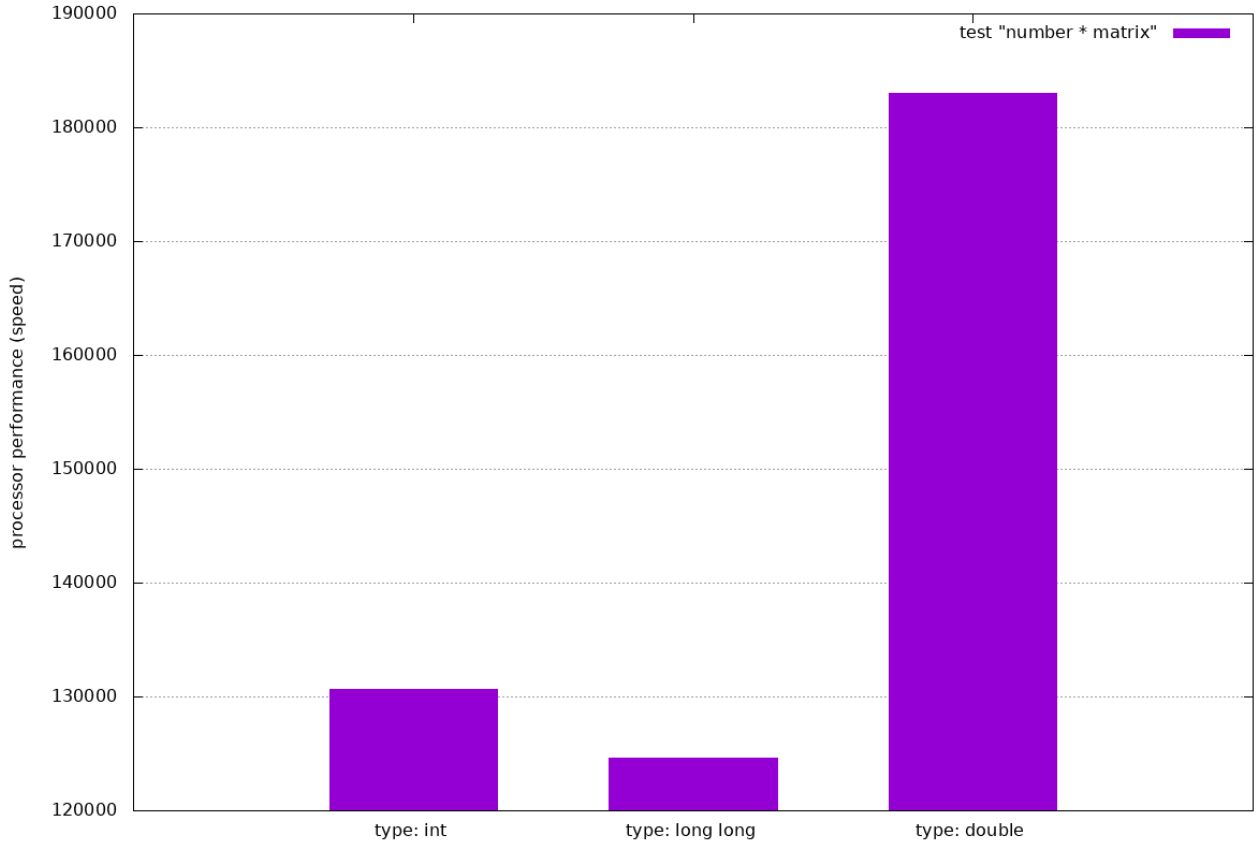

Рисунок 5: диаграмма производительности с ключом -O1

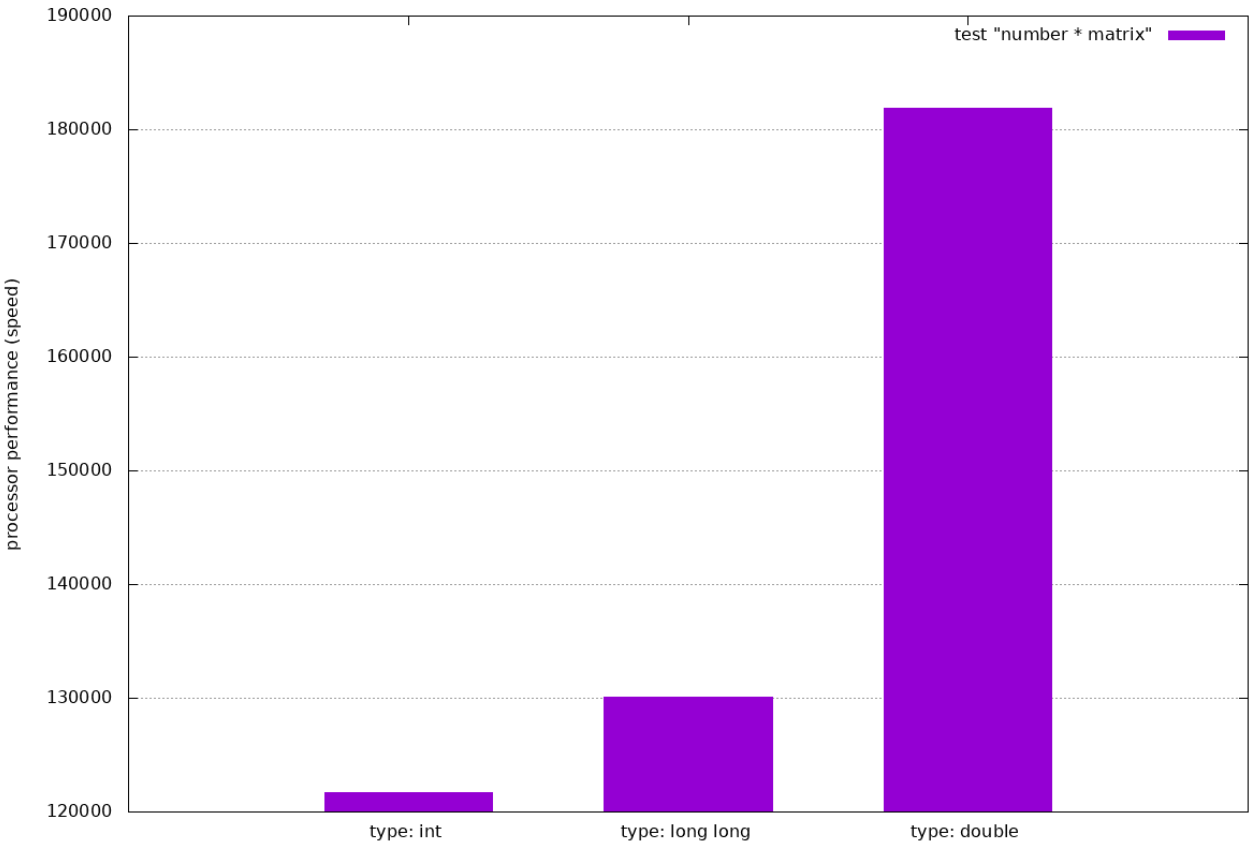

Рисунок 6: диаграмма производительности с ключом -O2

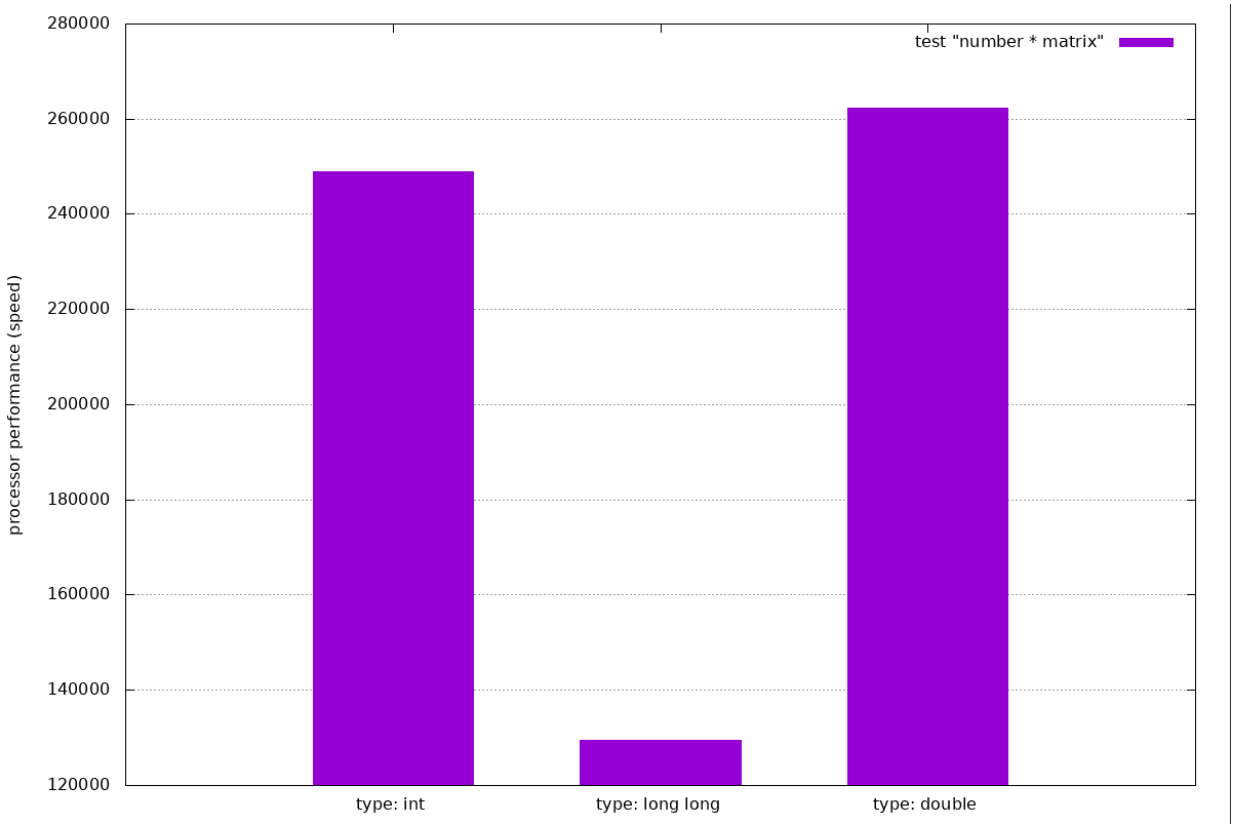

Рисунок 7: диаграмма производительности с ключом -O3

Приложение

Файл main.cpp

```
#include "foo.h"

int main() {
    srand(time(0));
    int cnt_tests = 3, i; ///количество тестов
    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]);
    }

    ///получение названия процессора
    char cpuname[50] = {'\0'};
    get_cpu_name(cpuname);
    //printf("%s %d \n", cpuname, (int)sizeof(cpuname));

    ///запись в output.csv
    FILE *fout;
    if ((fout = fopen("output.csv", "w")) == NULL) {
        printf("Can't open output.csv \n");
        return 1;
    }
                                                                    //fprintf(fout,
"PModel;Task;OpType;Opt;LNum;InsCount;Timer;AvTime;AbsErr;RelErr;TaskPerf\n");
    for (i = 0; i < cnt_tests; i++) {
        fprintf(fout, cpuname); ///proc
        fprintf(fout, ";");
        switch (i) {
            case 0:
                fprintf(fout, "num * matrix;int;None;");
                break;
            case 1:
                fprintf(fout, "num * matrix;long long;None;");
                break;
            case 2:
                fprintf(fout, "num * matrix;double;None;");
                break;
            default:
                printf("ERROR: wrong \"num\" while writing a file \n");
                break;
        }
        fprintf(fout, "%lld;", n[i]); ///lanch number
        switch (i) { ///instruction count
            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();"); ///Timer
        fprintf(fout, "%g;", avg_time[i]);
        fprintf(fout, "%g;", abs_error[i]);
        fprintf(fout, "%g%%;", rel_error[i] * 100); ///относительная погрешность
в %
        fprintf(fout, "%g;", 1 / avg_time[i]);
        fprintf(fout, "\n");
    }


    fclose(fout);
    return 0;
}
```

Файл foo.h

```
#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
```

Файл foo.cpp

```
#include "foo.h"

void test1_1(int mx, double &time) { ///типовая задача (тест)
    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) { ///типовая задача с другим типом данных
```

```
    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) { ///типовая задача с другим типом данных
    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; ///время выполнения только типовой задачи

    for (long long i = 0; i < n; i++) {
        switch (num) { ///выбор типовой задачи (теста)
            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;
        } ///конец switch
        summand1 += x * x; ///первое слагаемое для дисперсии
        summand2 += x; ///2-ое слагаемое
    }
    summand1 /= n;
    summand2 /= n;
    avg_time = summand2; ///sum(x[i]) / n == среднее время
    summand2 *= summand2;
    dispersion = summand1 - summand2; ///дисперсия (точность измерения времени)
        abs_error  =  sqrt(dispersion);  ///среднее  квадратическое  отклонение
(погрешность)
    rel_error = dispersion / avg_time; ///относительная погрешность
}

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

#изображение, где будет диаграмма
set terminal png font "Verdana,12" size 1200, 800
set output "diagram.png"

#символ-раделитель в outout.csv
set datafile separator ';'

# ось игрек
set ylabel "processor performance (speed)"

#установки сетки по одной оси
set grid ytics

#область значений для осей
#set yrange [0:400000]
```

```
set xrange [-1:3]

#подписи по оси абсцисс
set xtics ("type: int" 0, "type: long long" 1, "type: double" 2,)

#устаносить стиль диаграммы
set style data boxes

#установить ширину столбцов 0.6 от максимальной
set boxwidth 0.6 absolute
set style fill solid 1

#считать информацию
plot "output.csv" using 11 title "test \"number * matrix\""
```