Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

Звіт
з лабораторної роботи № 1
на тему:

«Визначення швидкодії обчислювальної системи»

Студента другого курсу
групи К-23(2)
Міщука Романа Андрійовича
Факультету комп’ютерних наук
та кібернетики

Київ – 2022

1. Постановка задачі

Необхідно розробити програму, яка вимірює кількість виконуваних базових операцій (команд) за секунду конкретною ОбСист (комп'ютер + ОС + Система програмування). Вимірювання "чистої" команди процесора не потрібне (як і є у реальних програмних комплексах, що типово розробляються на мовах високого рівня, часто навіть на платформенно незалежних) і фактично не має сенсу. Вибір системи програмування за критерієм "яка з них генерує швидший код" зайва, - виберіть ту з них, яка для вас найбільш зручна.

2. Використані методи

2.1. Різновиди тестів

Для тестування було обрано операції *додавання, віднімання, множення* та *ділення* для таких типів як: `int`, `long`, `long long`, `char`, `float`, `double`.

2.2. Проведення замірів

Для вимкнення оптимізації компілятором використане ключове слово `volatile`, що не дозволяло скорочувати повторення коду. Також, задля коректної роботи на деяких компіляторах, функція, що містить основний код повторень, обгорнута у відповідний набір макросів:

```
#pragma optimize( "", off )
#pragma GCC push_options
#pragma GCC optimize ("O0")

...

#pragma GCC pop_options
#pragma optimize( "", on )
```

Повторення коду досягалося рекурсією на рівні компіляції (за допомогою ключового слова `inline`), та також за допомогою звичайних циклів. Для того, щоб бути впевненим, що код справді був повторений під час компіляції, був використаний макрос:

```
#if defined(__GNUC__) || defined(__GNUG__)
#define in_void inline void __attribute__((always_inline))
#else
#define in_void __forceinline void
#endif
```

Заміри часу проводилися за допомогою бібліотеки `std::chrono`, та її вбудованого методу `high_resolution_clock::now()`.

2.3. Кросплатформенність

За допомогою інструменту *CMake* була реалізована можливість легкої компіляції коду під різні платформи та компплілятори.

Також за використання інструмента *Docker* робота програми була протестована на віртуальній машині під керуванням операційної системи Ubuntu.

3. Демонстрація роботи

Характеристики комп’ютера, використаного для тестування:

- Операційна система: Windows 11 x64;
- Процесор: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz;
- Компілятор: Mingw64-Release.

Результат роботи програми під час запуску в головній системі комп’ютера:

```
2.4975e+09
+ int       1.67112e+09     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                                67%
- int       1.71468e+09     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                               69%
* int       2.42954e+09     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    97%
/ int       3.56201e+08     XXXXXXX                                                          14%

+ long      8.41326e+08     XXXXXXXXXXXXXXXXX                                                34%
- long      1.22249e+09     XXXXXXXXXXXXXXXXXXXXXXX                                          49%
* long      1.64366e+09     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                               66%
/ long      4.50045e+08     XXXXXXXXX                                                        18%

+ llong     8.34307e+08     XXXXXXXXXXXXXXXXX                                                33%
- llong     1.24906e+09     XXXXXXXXXXXXXXXXXXXXXXXXX                                        50%
* llong     7.14082e+08     XXXXXXXXXXXXXX                                                   29%
/ llong     3.22601e+08     XXXXXX                                                           13%

+ char      2.4975e+09      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    100%
- char      8.20749e+08     XXXXXXXXXXXXXXXX                                                 33%
* char      1.20453e+09     XXXXXXXXXXXXXXXXXXXXXXX                                          48%
/ char      4.52899e+08     XXXXXXXXX                                                        18%

+ float     1.22011e+09     XXXXXXXXXXXXXXXXXXXXXXX                                          49%
- float     1.28304e+09     XXXXXXXXXXXXXXXXXXXXXXXXX                                        51%
* float     1.01647e+09     XXXXXXXXXXXXXXXXXXX                                              41%
/ float     1.26295e+09     XXXXXXXXXXXXXXXXXXXXXXXXX                                        51%

+ double    1.71292e+09     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                               69%
- double    1.67504e+09     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                               67%
* double    9.91473e+08     XXXXXXXXXXXXXXXXXXX                                              40%
/ double    5.09528e+08     XXXXXXXXXX                                                       20%
```

Результат роботи програми у віртуальному середовищі Docker під керуванням операційної системи Ubuntu (компілятор GCC, середовище запущене на тому ж комп’ютері):

```
2.77121e+08
+ int      2.23098e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX            81%
- int      2.56086e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX     92%
* int      2.56229e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX     92%
/ int      2.59154e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    94%

+ long     2.69418e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  97%
- long     1.59385e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXX                          58%
* long     2.43425e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             88%
/ long     2.44797e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             88%

+ llong    2.48866e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             90%
- llong    2.77121e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  100%
* llong    2.70267e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    98%
/ llong    2.46716e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             89%

+ char     2.05071e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                     74%
- char     2.65877e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX          96%
* char     2.69362e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX       97%
/ char     2.62953e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX          95%

+ float    2.67815e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX         97%
- float    2.60259e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             94%
* float    2.59863e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             94%
/ float    2.49151e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX               90%

+ double   2.60073e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             94%
- double   2.65449e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX           96%
* double   2.62868e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             95%
/ double   2.38984e+08     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                86%
```

4. Код програми

4.1. Dockerfile

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y g++ cmake make

COPY . /usr/src/LAB1
WORKDIR /usr/src/LAB1

RUN cmake -S . -B ./build
RUN cmake --build ./build

CMD ./build/main
```

4.2. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15)
project(LAB1_AES VERSION 1.0 LANGUAGES CXX)

set(CMAKE_CXX_FLAGS_RELEASE "-O2")

add_executable(main LAB1.cpp)
```

4.3. LAB1.cpp

```
#include <iostream>
#include <chrono>
#include <tuple>
#include <vector>
#include <string>
#include <iomanip>
#include <cmath>

using namespace std;

#if defined(__GNUC__) || defined(__GNUG__)
#define in_void inline void __attribute__((always_inline))
#else
#define in_void __forceinline void
#endif

enum operation
{
	addition,
	subtraction,
	multiplication,
	division,
	no_operation
};
template <operation T>
using operation_t = integral_constant<operation, T>;
constexpr operation_t<no_operation> no_operation_constant{};

template <typename T>
using op_arg_t = volatile T;

template<typename T>
in_void exec_operation(op_arg_t<T> a, op_arg_t<T> b, volatile T& res, operation_t<addition>)
{ res = a + b; }
template<typename T>
in_void exec_operation(op_arg_t<T> a, op_arg_t<T> b, volatile T& res,
operation_t<subtraction>)
{ res = a - b; }
template<typename T>
in_void exec_operation(op_arg_t<T> a, op_arg_t<T> b, volatile T& res,
operation_t<multiplication>)
{ res = a * b; }
template<typename T>
in_void exec_operation(op_arg_t<T> a, op_arg_t<T> b, volatile T& res, operation_t<division>)
{ res = a / b; }
template<typename T>
in_void exec_operation(op_arg_t<T> a, op_arg_t<T> b, volatile T& res,
operation_t<no_operation>)
{
	res = a;
	//res = b;
}



typedef unsigned long long _loop_t;
enum base_using { yes, no };
template <base_using T>
using base_using_t = integral_constant<base_using, T>;

template <typename T>
using rep_arg_t = volatile T&;

template<typename T, operation O, const _loop_t R, enable_if_t<0 == R, bool> = 0>
```

```
in_void exec_repeated(rep_arg_t<T> a, rep_arg_t<T> b, volatile T& res, const operation_t<O>&
op, const base_using_t<no>)
{
	exec_operation(a, b, res, op);
}
template<typename T, operation O, const _loop_t R, enable_if_t<0 == R, bool> = 0>
in_void exec_repeated(rep_arg_t<T> a, rep_arg_t<T> b, volatile T& res, const operation_t<O>&
op, const base_using_t<yes>)
{
	exec_operation(a, b, res, op);
	exec_operation(a, b, res, no_operation_constant);
}
template<typename T, operation O, const _loop_t R, base_using B, enable_if_t<0 < R, bool > =
0>
in_void exec_repeated(rep_arg_t<T> a, rep_arg_t<T> b, volatile T& res, const operation_t<O>&
op, const base_using_t<B>& base)
{
	exec_repeated<T, O, 0>(a, b, res, op, base);
	exec_repeated<T, O, R-1, B>(a, b, res, op, base);
}

template<typename T, operation O, const _loop_t R, base_using B>
double measure_time(rep_arg_t<T> a, rep_arg_t<T> b, volatile T& res, const operation_t<O>&
op, const base_using_t<B>& base)
{
	chrono::high_resolution_clock::time_point begin{}, end{};

	begin = chrono::high_resolution_clock::now();
	exec_repeated<T, O, R>(a, b, res, op, base);
	end = chrono::high_resolution_clock::now();

	return chrono::duration_cast<chrono::nanoseconds>(end - begin).count();
}


#pragma optimize( "", off )
#pragma GCC push_options
#pragma GCC optimize ("O0")
template <typename T, operation O, const _loop_t R>
double run_test(const _loop_t count = 100)
{
	constexpr base_using_t<yes> op_based{};
	constexpr base_using_t<no> op_not_based{};

	const operation_t<O> op{};
	double ret = 0, buff;
	T res = 0, a = 1, b = 1;
	for (_loop_t i = 0; i < count; i++) {
		buff = 2 * measure_time<T, O, R>(a, b, res, op, op_not_based) -
			measure_time<T, O, R>(a, b, res, op, op_based);
		if(buff < 0)
		{
			i--;
			continue;
		}
		ret += buff;
	}
	return R * count * 1e9 / ret;
}
#pragma GCC pop_options
#pragma optimize( "", on )


template<typename T, const _loop_t R, const _loop_t C>
tuple<string, double> run_test_for(const operation O)
{
	switch (O)
	{
```

```
	case addition:
		return {"+", run_test<T, addition, R>(C)};
	case subtraction:
		return {"-", run_test<T, subtraction, R>(C)};
	case multiplication:
		return {"*", run_test<T, multiplication, R>(C)};
	case division:
		return {"/", run_test<T, division, R>(C)};
	default:
		return {"", 0};
	}
}

template<const _loop_t R, const _loop_t C>
tuple<string, string, double> run_test_for(const int type_id, const operation O)
{
	switch (type_id)
	{
	case 0:
		return tuple_cat(make_tuple("int"), run_test_for<int, R, C>(O));
	case 1:
		return tuple_cat(make_tuple("long"), run_test_for<long, R, C>(O));
	case 2:
		return tuple_cat(make_tuple("llong"), run_test_for<long long, R, C>(O));
	case 3:
		return tuple_cat(make_tuple("char"), run_test_for<char, R, C>(O));
	case 4:
		return tuple_cat(make_tuple("float"), run_test_for<float, R, C>(O));
	case 5:
		return tuple_cat(make_tuple("double"), run_test_for<double, R, C>(O));
	default:
		return tuple_cat(make_tuple(""), run_test_for<int, R, C>(O));
	}
}

void print_result(const double max, const tuple<string, string, double>& result)
{
	constexpr int bar_len = 50;

	cout << setw(2) << std::left << std::get<1>(result)
		<< setw(10) << std::left << std::get<0>(result)
		<< setw(16) << std::left << std::get<2>(result)
		<< setw(bar_len + 1) << std::left << string(round(std::get<2>(result) *
bar_len / max), 'X')
		<< round(std::get<2>(result) * 100 / max) << "%\n";
}

constexpr int types_num = 6, ops_num = 4;
void take_measurements(vector<vector<tuple<string, string, double>>> &results, double& max,
double& avg)
{
	constexpr _loop_t R = 100, C = 1e5;
	constexpr double infty = numeric_limits<double>::infinity();

	max = avg = 0;
	results = vector<vector<tuple<string, string, double>>>(types_num,
vector<tuple<string, string, double>>(ops_num));

	tuple<string, string, double> buff_result{};
	double& time = std::get<2>(buff_result);
	for (int type = 0; type < types_num; type++)
		for (int op = 0; op < ops_num; op++) {
			time = 0;
			while (time <= 0 || !(time < infty)) {
				buff_result = run_test_for<R, C>(type, (operation)op);
			}
			results[type][op] = buff_result;
```

```
                        if (time > max) max = time;
                        avg += time;
                }
        avg /= types_num * ops_num;
}

int main()
{
        cout.precision(6);
        double max = 1e9, avg = 1;
        vector<vector<tuple<string, string, double>>> results;

        // Running tests till system is stable
        while (max / avg > 5) {
                take_measurements(results, max, avg);
        }

        cout << max << "\n";
        for (int type = 0; type < types_num; type++) {
                for (int op = 0; op < ops_num; op++)
                        print_result(max, results[type][op]);
                cout << "\n";
        }
        return 0;
}
```