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

# 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.

# Використані методи

## Різновиди тестів

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

## 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().

## Кросплатформенність

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

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

# 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 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 67%

- int 1.71468e+09 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 69%

\* int 2.42954e+09 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 97%

/ int 3.56201e+08 XXXXXXX 14%

+ long 8.41326e+08 XXXXXXXXXXXXXXXXX 34%

- long 1.22249e+09 XXXXXXXXXXXXXXXXXXXXXXXX 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 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 100%

- char 8.20749e+08 XXXXXXXXXXXXXXXX 33%

\* char 1.20453e+09 XXXXXXXXXXXXXXXXXXXXXXXX 48%

/ char 4.52899e+08 XXXXXXXXX 18%

+ float 1.22011e+09 XXXXXXXXXXXXXXXXXXXXXXXX 49%

- float 1.28304e+09 XXXXXXXXXXXXXXXXXXXXXXXXXX 51%

\* float 1.01647e+09 XXXXXXXXXXXXXXXXXXXX 41%

/ float 1.26295e+09 XXXXXXXXXXXXXXXXXXXXXXXXX 51%

+ double 1.71292e+09 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 69%

- double 1.67504e+09 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 67%

\* double 9.91473e+08 XXXXXXXXXXXXXXXXXXXX 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 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 92%

\* int 2.56229e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 92%

/ int 2.59154e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 94%

+ long 2.69418e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 97%

- long 1.59385e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXX 58%

\* long 2.43425e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 88%

/ long 2.44797e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 88%

+ llong 2.48866e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 90%

- llong 2.77121e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 100%

\* llong 2.70267e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 98%

/ llong 2.46716e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 89%

+ char 2.05071e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 74%

- char 2.65877e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 96%

\* char 2.69362e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 97%

/ char 2.62953e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 95%

+ float 2.67815e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 97%

- float 2.60259e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 94%

\* float 2.59863e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 94%

/ float 2.49151e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 90%

+ double 2.60073e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 94%

- double 2.65449e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 96%

\* double 2.62868e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 95%

/ double 2.38984e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 86%

# Код програми

## 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

## 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)

## 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;

}