

Міністерство освіти і науки України Національний технічний університет України

«Київський політехнічний інститут»

Лабораторна робота №4

**ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ ЗА ДОПОМОГОЮ STL**

Виконав студент групи: КВ-03

ПІБ: Жовтанюк М.В

**Київ 2022**

**Паралельне програмування за допомогою STL**

Завдання:

**Частина 1**

Реалізувати паралельну обробку елементів масиву відповідно до варіанту,

використовуючи механізм асинхронних викликів (std::async). Для цього

розбити масив на M однакових частин та виконувати обробку кожної з них

окремо. Масив початково заповнюється випадковими значеннями.

Виміряти час обробки масиву для декількох різних значень M (наприклад,

1, 2, 3, 4, 8, 16), а також порівняти його для випадків послідовної та

паралельної обробки (використовувати політики запуску std::launch::async та

std::launch::deferred). Результати дослідження представити у вигляді таблиці у

звіті. Бажано також у звіті представити короткі висновки з аналізу отриманих

результатів.

Розмір масиву обрати на власний розсуд в залежності від конфігурації

апаратного забезпечення системи (орієнтуватися на час роботи не менший

ніж 10 мс для забезпечення точності вимірів). Тип елементів масиву обрати

також самостійно.

В якості додаткового завдання – реалізувати можливість задання розміру

масиву, кількості частин (для розбиття) та політики запуску користувачем.

Номер варіанту визначається відповідно до номеру за списком за

наступною формулою:

**Варіанти завдань:**

2. Розрахунок суми значень елементів масиву

константне значення (з перезаписом значень масиву)

**Частина 2**

Реалізувати систему, що зображена на рисунку.

Система містить наступні сутності: N «генераторів» та один «процесор».

Генератор генерує випадкове значення та передає його процесору, після чого

«засинає» на випадковий проміжок часу (діапазон значень обрати самостійно).

Процесор чекає значення від генераторів та виводить їх на екран (виводячи

також ідентифікатор (наприклад, порядковий номер) генератора). Кожна з

сутностей (як генератори, так і процесор) має виконуватися в окремому потоці.

Реалізувати механізм завершення роботи програми (наприклад, шляхом

натиснення деякої клавіші).

В якості додаткового завдання може бути реалізовано, наприклад,

процедури додавання/видалення генераторів, а також конфігурування

кожного з них (діапазон часу затримки, діапазон генерованих значень).

Код програми:

main.cpp

#include <iostream>

#include "async.h"

void part\_1();

void part\_2();

std::vector<long> async\_vector;

std::vector<std::future<long>> res\_vector;

int VECTOR\_SIZE = 100000000;

int parts\_amount = 0;

int l\_policy\_option = 0;

using namespace std::chrono;

int main() {

int option = 2;

std::cout << "1. First part, time measurement" << std::endl;

std::cout << "2. Second part, processor-generator" << std::endl;

std::cout << '**\n**';

switch (option) {

case 1:

part\_1();

break;

case 2:

part\_2();

break;

default:

break;

}

return 0;

}

void part\_1() {

std::cout << "Enter array size: ";

std::cin >> VECTOR\_SIZE;

std::cout << '**\n**';

std::launch l\_policy;

std::cout << "Please enter amount of parts: ";

std::cin >> parts\_amount;

std::cout << "Press 1 to launch in deferred policy or 2 for async: ";

std::cin >> l\_policy\_option;

if (l\_policy\_option == 1) l\_policy = std::launch::deferred;

else if (l\_policy\_option == 2) l\_policy = std::launch::async;

else {

std::cout << "Wrong option, terminating..." << std::endl;

exit(1);

}

async\_vector.resize(VECTOR\_SIZE);

for (auto &i: async\_vector) {

i = random() % VECTOR\_SIZE;

}

std::vector<std::vector<long>> parts;

parts.resize(parts\_amount);

int one\_part\_amount = VECTOR\_SIZE / parts\_amount;

for (int i = 0, j = 0; i < async\_vector.size(); i++) {

parts[j].push\_back(async\_vector[i]);

if (i == one\_part\_amount) j++;

}

// async

for (int i = 0; i < parts.size(); i++) {

res\_vector.push\_back(std::async(l\_policy, sum\_calc, parts[i], i));

}

long res\_sum = 0;

// start

auto start = steady\_clock::now();

for (auto &i: res\_vector) {

res\_sum += i.get();

}

auto stop = steady\_clock::now();

// stop

auto time\_res = duration\_cast<std::chrono::milliseconds>(stop - start);

auto ms = time\_res.count();

std::cout << "Time: " << ms << std::endl;

std::cout << "Sum result: " << res\_sum << std::endl;

}

void part\_2() {

std::thread th\_proc(processor);

th\_proc.join();

}

async.h

//

// Created by Maksym Zhovtaniuk on 19.06.2022.

//

#ifndef OOP\_LAB4\_ASYNC\_H

#define OOP\_LAB4\_ASYNC\_H

#include <future>

#include <vector>

#include <cstdlib>

#include <string>

#include <chrono>

#include <sstream>

std::mutex m;

static bool exit\_status;

long sum\_calc(const std::vector<long> &v, const int tag) {

long res = 0;

for (auto &i: v) {

res += i;

}

return res;

}

void generator(std::vector<std::vector<long>> &v, const int tag) {

while (!exit\_status) {

m.lock(); // lock mutex

long random\_num = random();

long ms = 1000 + random() % (5000 - 1000 + 1);

std::ostringstream oss;

oss << "Thread" << tag << " returns random number = " << random\_num << " and sleeps for " << ms

<< " milliseconds";

std::string str = oss.str();

std::cout << str << std::endl;

std::vector<long> res = {tag, random\_num, ms};

v.push\_back(res);

m.unlock(); // unlock mutex

std::this\_thread::sleep\_for(std::chrono::milliseconds(ms));

}

}

void processor() {

exit\_status = false;

int generators\_num = 3;

std::vector<std::thread> threads\_generators;

threads\_generators.resize(generators\_num);

std::vector<std::vector<long>> random\_res;

for (int i = 0; i < generators\_num; i++) {

threads\_generators[i] = std::thread(&generator, std::ref(random\_res), i);

}

std::cin.get();

exit\_status = true;

for (auto &i: threads\_generators) {

i.join();

}

std::cout << "All threads has been finished" << std::endl;

std::cin.get();

}

#endif //OOP\_LAB4\_ASYNC\_H

* Посилання на [github-репозиторій](https://github.com/zh0vtyj/oop-lab4)
* Також у проекті використано CMake

Принцип роботи:

1. Користувач задає кількість частин та розмір масиву, далі цей вектор заповнюється випадковими значеннями у діапазоні

[0, розмір\_масиву], створюється вектор векторів з довжиною

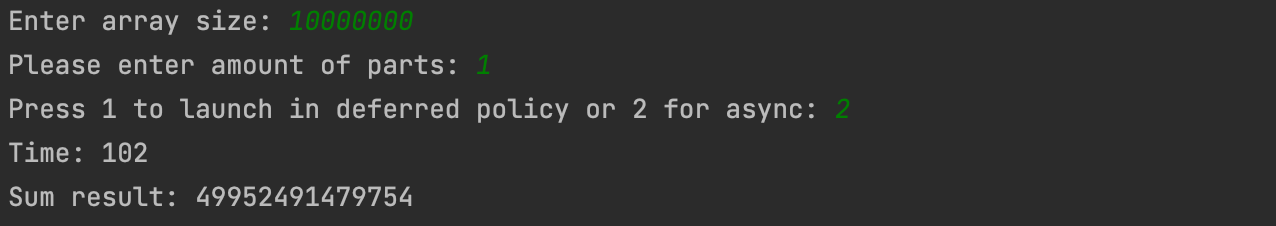
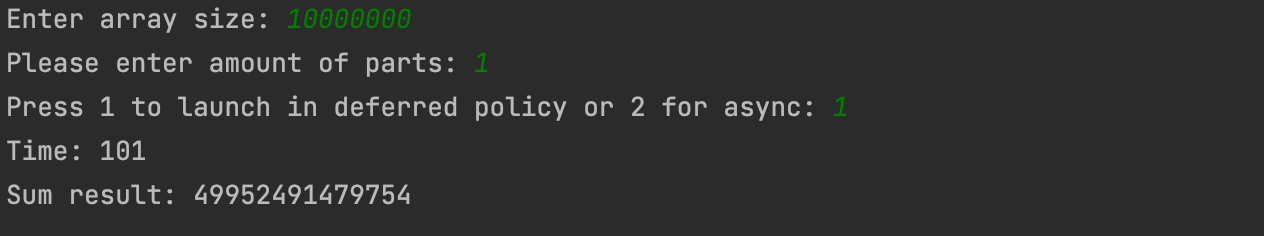
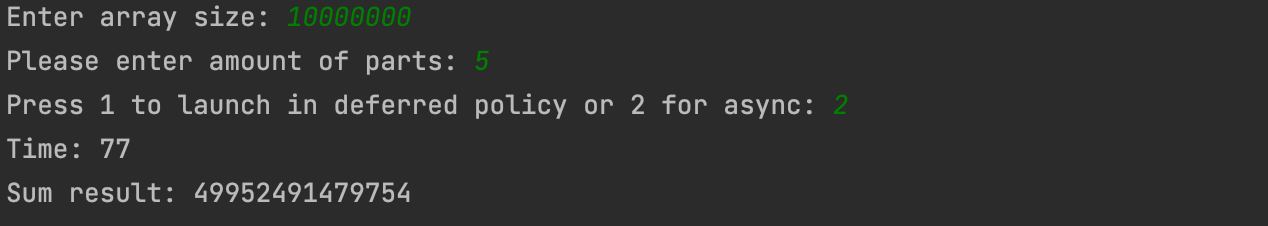
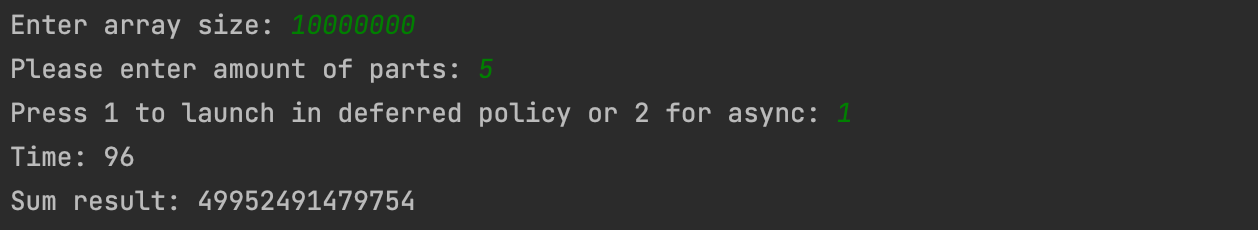
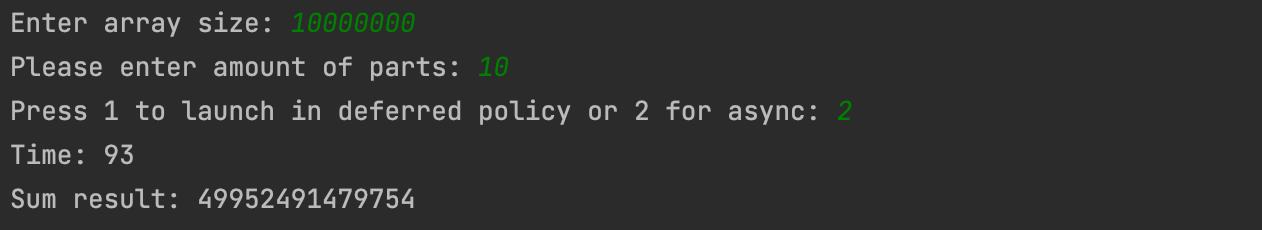
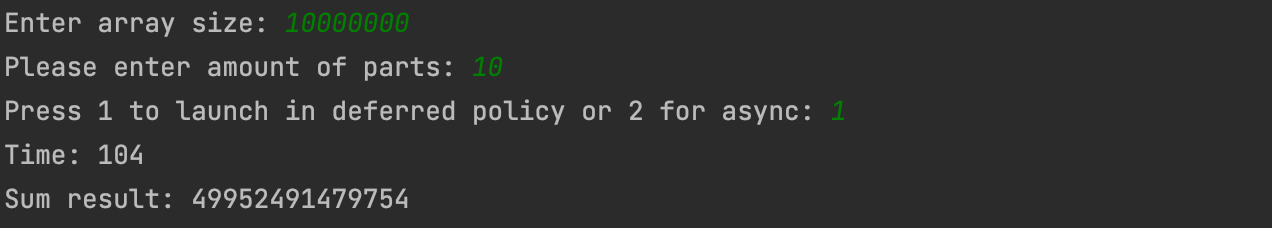
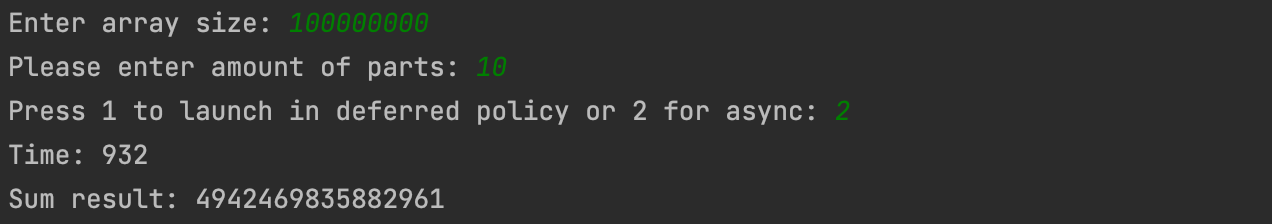
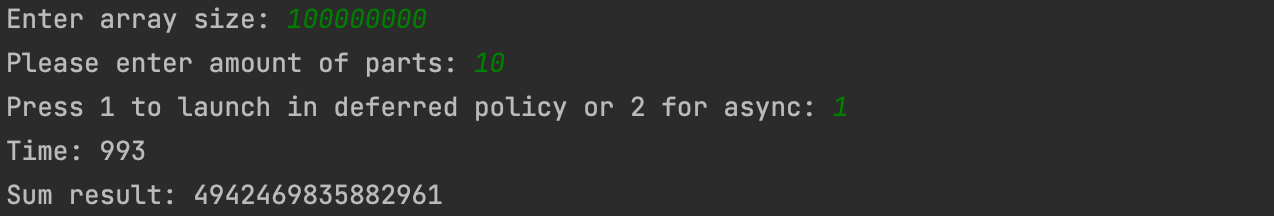
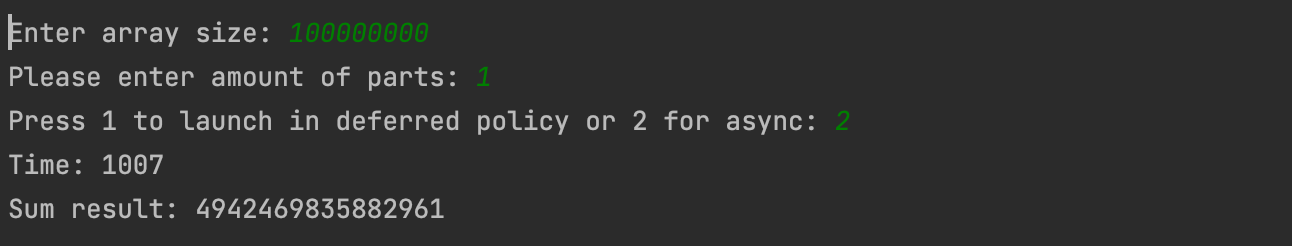
N – кількість частин (по суті матриця), кожен вкладений вектор буде передаватися як параметр у потік, елементи цього масиву заповнюється відповідними значеннями із початкового. Створюється вектор ф’ючерсів, де параметрами у std::async

передаються: політика виклику (послідовна чи асинхронна), вказівник на функцію, частина, яку буде обробляти потік та порядковий номер потоку. Далі проводяться заміри часу та виконуються асинхронні виклики.

1. Створюється потік-процесор, в якому користувач спочатку вказує кількість генераторів, потім створюється вектор потоків та вектор результатів, який містить: номер потоку, випадкове число, що потік створить та випадковий час затримки після кожної ітерації, далі створюються потоки, у які передаються: функція, що виконуватиметься, вказівник на вектор, у який будуть заноситись результати кожної ітерації та номер потоку. Процесор чекає поки користувач натисне ENTER, а потім закінчує потоки. Генератори-потоки працюють поки користувач не вийде (використовується булева змінна, яка міняє стан по натисненні на ENTER), спочатку блокується раніше ініціалізований м’ютекс, щоб ніхто не «вліз» у роботу потока, генерується випадкове число та випадкове значення затримки в діапазоні [1000, 5000] (від 1 до 5 секунд), потік виводить результати та заносить у вектор результатів значення кожної ітерації, далі м’ютекс розблоковується і потік «засинає» на визначену раніше випадкову величину.

**Тести:**

**Частина 1:**



\*Усі тести нижче розміру масиву 1000000 дають нуль

Кількість частин M = 10

|  |  |  |
| --- | --- | --- |
| **Розмір масиву** | **Deferred (послідовний)** | **Async (асинхронний)** |
| 1000000 | 10 | 9 |
| 10000000 | 96 | 92 |
| 100000000 | 1080 | 911 |

Кількість частин M = 5

|  |  |  |
| --- | --- | --- |
| **Розмір масиву** | **Deferred (послідовний)** | **Async (асинхронний)** |
| 1000000 | 10 | 8 |
| 10000000 | 95 | 77 |
| 100000000 | 999 | 819 |

Кількість частин M = 1

|  |  |  |
| --- | --- | --- |
| **Розмір масиву** | **Deferred (послідовний)** | **Async (асинхронний)** |
| 1000000 | 10 | 10 |
| 10000000 | 96 | 101 |
| 100000000 | 1012 | 1007 |

**Висновок:**

У загальному випадку видно, що зі збільшенням розміру масиву, швидкість роботи асинхронного режиму зменшується час виконання задачі обчислення суми, зменшення ж однакових частин масиву, поводить себе непередбачувано, то збільшуючи, то зменшуючи час роботи.

**Частина 2:**

