МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”
Кафедра систем штучного інтелекту

Лабораторна робота
з дисципліни
« Технології розподілених систем та паралельних обчислень»

Виконав:
студент групи КН-309
Ляшеник Остап
Викладач:
Мочурад Л. І.

2024 р.

Лабораторна робота № 3

Тема: Завантаження та синхронізація в OpenMP

Мета: Вивчити роботу з операторами керування паралельними потоками засобами OpenMP

7	$\int_{4}^{6} \frac{dx}{x \cdot (x^2 + 3{,}5^2)}$	$\int_{0{,}6}^{1{,}6} \frac{dx}{1 + \cos(0{,}8 \cdot x) + \sin(0{,}8 \cdot x)}$

Код

```
import concurrent.futures
import sympy as sp
import time
import threading
import os

# Налаштування замка
lock = threading.Lock()
file_name = "integration_results.txt"

def left_rectangle_approximation(expression, variable, lower_bound,
upper_bound, num_rectangles):
    """
    Calculate the approximate value of an integral using the left
rectangle method.

    :param expression: sympy expression to integrate
    :param variable: sympy symbol, integration variable
    :param lower_bound: float, lower bound of the integral
    :param upper_bound: float, upper bound of the integral
    :param num_rectangles: int, number of rectangles to use in the
approximation
    :return: float, approximate value of the integral
    """
    # Calculate the width of each rectangle
    delta_x = (upper_bound - lower_bound) / num_rectangles
    # Calculate the x values at the left end of each rectangle
    x_values = [lower_bound + i * delta_x for i in range(num_rectangles)]
    # Calculate the area of each rectangle and sum them up
    total_area = sum(expression.subs(variable, x_val) for x_val in
x_values) * delta_x
    return total_area

def integrate_function(expression, variable, bounds, step_size):
    num_rectangles = 100 #int((bounds[1] - bounds[0]) / step_size)
    result = left_rectangle_approximation(expression, variable, bounds[0],
```

```
bounds[1], num_rectangles)
    return result

def thread_function(name, expression, variable, bounds, step_size):
    with lock:
        with open(file_name, "a") as f:
            f.write(f"The beginning of the closed section by thread
{name}...\n")
    # Розрахунок інтегралу
    result = integrate_function(expression, variable, bounds, step_size)
    with lock:
        with open(file_name, "a") as f:
            f.write(f"The end of the closed section by thread
{name}...\n")
    return result

x1 = sp.symbols('x')
expression1 = 1 / (x1 * (x1**2 + 3.5**2))
brakes_integral1 = [4,6]

x2 = sp.symbols('x')
# Define the expression for integral number 2
expression2 = 1 / (1 + sp.cos(0.8 * x2) + sp.sin(0.8 * x2))

brakes_integral2 = [0.6,1.6]

# Ваші інтеграли та кроки дискретизації
integrals = [
    (expression1, x1, brakes_integral1, 0.00002),
    (expression2, x2, brakes_integral2, 0.00002)
]

# Видаліть файл результатів, якщо він вже існує
if os.path.exists(file_name):
    os.remove(file_name)

# Стартові параметри для паралелізму
steps = [0.001, 0.0001, 0.00001, 0.00002]
num_cores = [1, 2, 4, 8]  # Залежно від вашого процесора
results = []

for step_size in steps:
    for cores in num_cores:
        # Вимірювання часу обчислень
        start_time = time.time()
        with concurrent.futures.ThreadPoolExecutor(max_workers=cores) as
executor:
            futures = [executor.submit(thread_function, f"Thread-{i}",
expr, var, bounds, step_size)
```

```
                    for i, (expr, var, bounds, _) in
enumerate(integrals)]
            for future in concurrent.futures.as_completed(futures):
                results.append(future.result())
        end_time = time.time()
        print(f"Step size: {step_size}, Cores: {cores}, Time taken:
{end_time - start_time}")

# Додавання результатів у файл
with open(file_name, "a") as f:
    for result in results:
        f.write(str(result) + "\n")
```

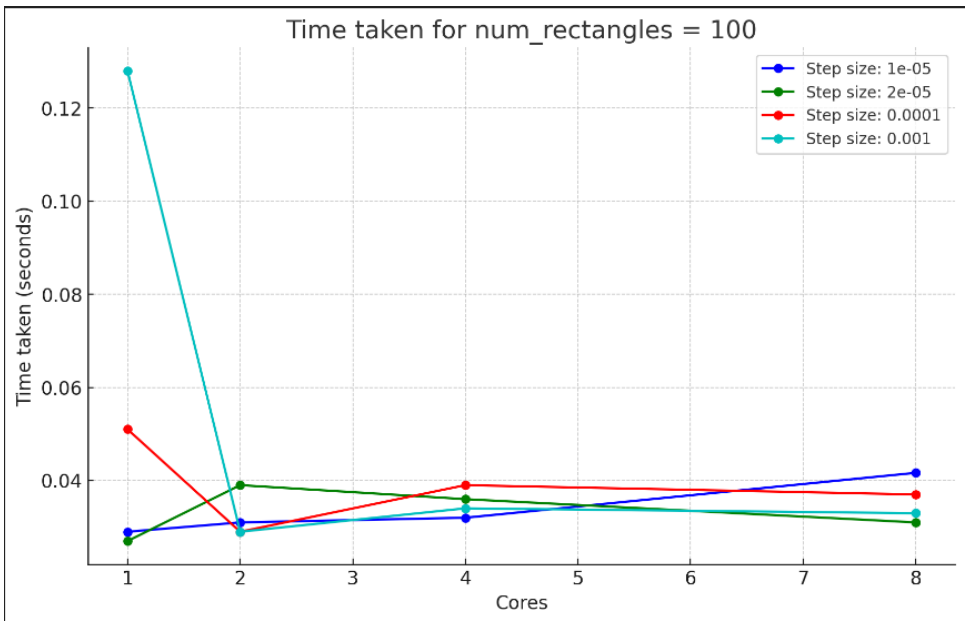
Аналіз результатів

Результат виконання, при **`num_rectangles = 100`**

Таблиця 1

Step size	Cores	Time taken
0.001	1	0.127955
0.0001	1	0.050998
0.00001	1	0.028996
0.00002	1	0.026998
0.001	2	0.028998
0.0001	2	0.029000
0.00001	2	0.031000
0.00002	2	0.039007
0.001	4	0.034022
0.0001	4	0.039004
0.00001	4	0.032004
0.00002	4	0.035995
0.001	8	0.032979
0.0001	8	0.037000
0.00001	8	0.041654
0.00002	8	0.031000

Рис 1.1 : графік до таблиці 1

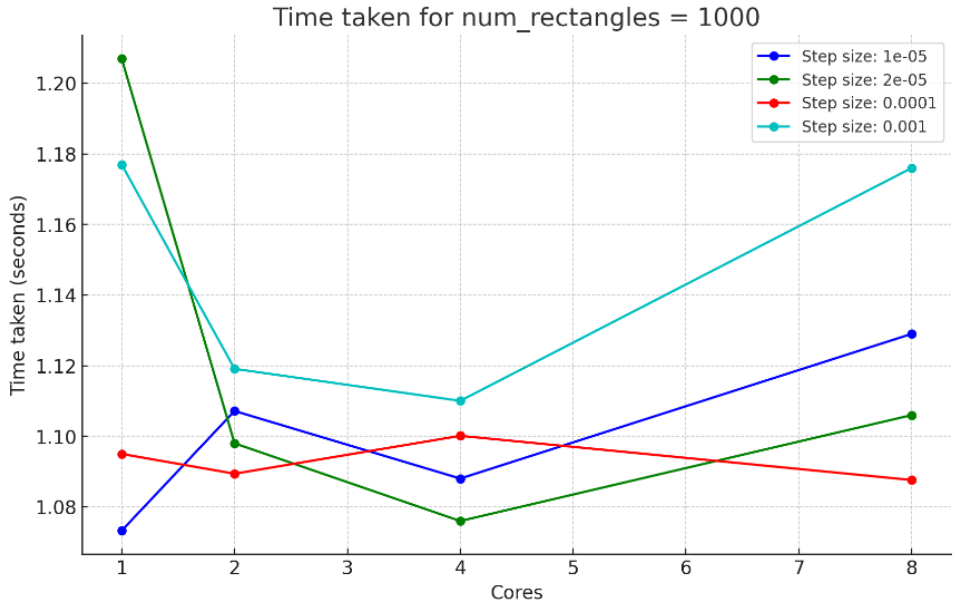


Результат виконання, при **`num_rectangles = 1000`**

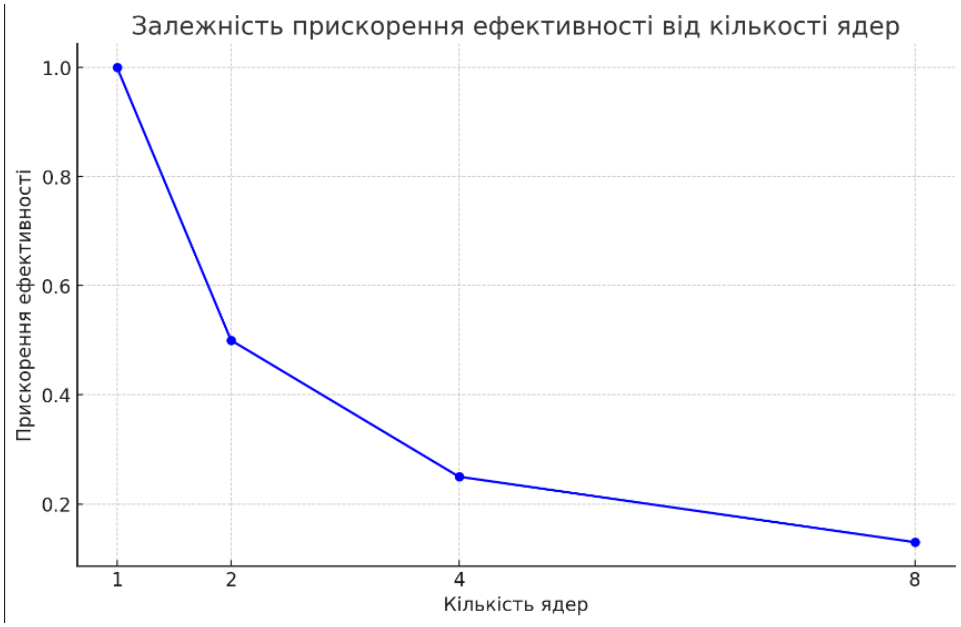
Таблиця 1.2

Step size	Cores	Time taken
0.001	1	1.176992
0.0001	1	1.095001
0.00001	1	1.073279
0.00002	1	1.206969
0.001	2	1.119102
0.0001	2	1.089388
0.00001	2	1.107162
0.00002	2	1.098004
0.001	4	1.110039
0.0001	4	1.100135
0.00001	4	1.088024
0.00002	4	1.075996
0.001	8	1.176001
0.0001	8	1.087612
0.00001	8	1.129007
0.00002	8	1.106003

Рис 1.2 : Графік до таблиці 1.2



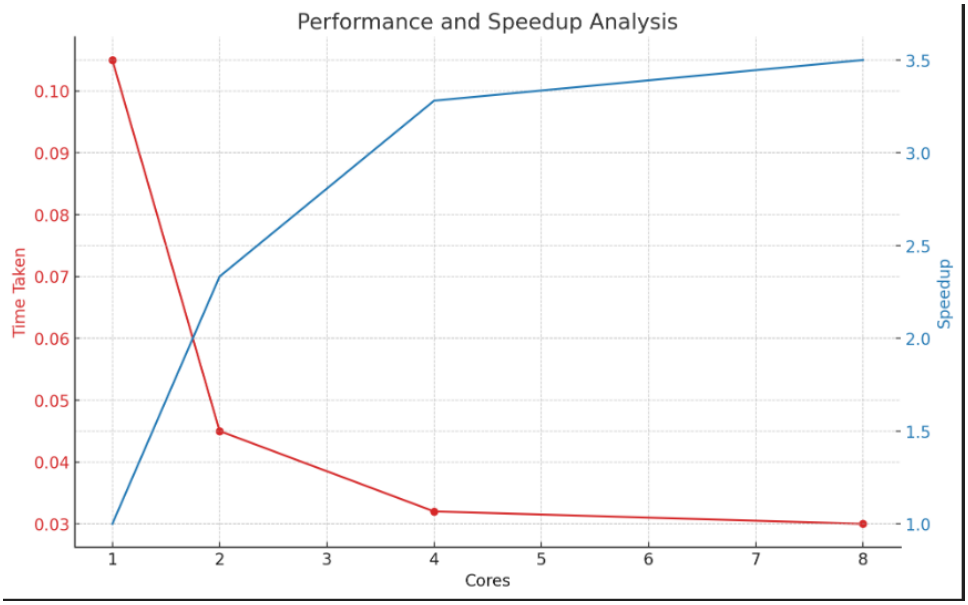
Кількість ядер	1	2	4	8
Прискорення	1.0	0.5	0.25	0,13



Таблиця 1.3 Прискорення

Step Size	Cores	Time Taken	Speedup
0.001	1	0.10500025749206543	1.000000
0.001	2	0.045000314712524414	2.3333227370261462
0.001	4	0.032003164291381836	3.2809336144407775
0.001	8	0.029996633529663086	3.5004013829829512

Графік 1.3 до таблиці 1.3



Аналізуючи надані дані та візуалізації, можна зробити наступні висновки:

- **Паралелізм значно покращує продуктивність:** Зі збільшенням кількості ядер (потоків) час обчислення зменшується, що демонструє ефективність паралелізму для даної задачі обчислення логарифмів.
- **Вплив розміру кроку на час обчислень:** Зменшення розміру кроку, яке зазвичай пов'язане з підвищенням точності обчислень, призводить до збільшення часу виконання. Це спостерігається в обох наборах даних (**num_rectangles = 100** і **num_rectangles = 1000**), підкреслюючи необхідність балансу між точністю та продуктивністю.
- **Масштабування задачі:** Збільшення обсягу задачі (з 100 до 1000 прямокутників) впливає на час виконання, але вплив паралелізму залишається важливим фактором для оптимізації продуктивності.
- **Оптимізація під конкретні параметри:** Вибір оптимальної кількості ядер та розміру кроку може значно варіюватися в залежності від конкретних

вимог до задачі (швидкість проти точності), що вимагає додаткової адаптації під кожний конкретний сценарій.

Висновок:

Отже, дані демонструють, що ефективне використання ресурсів обчислення (особливо паралелізм) може значно покращити продуктивність обчислень, але потрібно враховувати інші фактори, як-от розмір кроку та обсяг задачі, для досягнення оптимальних результатів.