Основи об'єктно-орієнтованого програмування – 2024
Лабораторна робота №3а
Паралельне мультипоточне програмування

Для цієї лабораторної роботи була обрана реалізація алгоритмів підрахунку кількості чисел масиву, оскільки саме цей варіант дозволяє досить добре продемонструвати відмінності продуктивності послідовного та паралельного виконань.

Структура:
У коді використовувалась стандартна бібліотека C++ для створення та керування потоками, а усі класи та функції було реалізовано з дотриманням принципів об'єктно-орієнтованого програмування та інкапсуляції.

- **main.cpp**: Цей файл містить головну функцію main, яка є входом у програму. В цьому файлі запускаються функції підрахунку чисел послідовно та паралельно та виводяться результати на екран.
- **count.h**: Цей файл містить декларації функцій для підрахунку чисел у масиві. В ньому оголошені функції countNumbersSequential та countNumbersParallel.
- **count.cpp**: Цей файл містить визначення функцій, оголошених у файлі count.h. В ньому містяться реалізації функцій для послідовного та паралельного підрахунку чисел у масиві.
- **CountTests.mm**: Цей файл містить юніт-тести для перевірки правильності роботи функцій підрахунку.

Результати виконання алгоритмів:

```
Послідовний підрахунок:
4: 4
3: 3
2: 2
1: 1
Час виконання послідовного підрахунку: 5.208e-06 секунд
Паралельний підрахунок:
2: 2
1: 1
3: 3
4: 4
Час виконання паралельного підрахунку: 0.000151875 секунд
```

Обидві версії функцій підрахунку працюють з одним і тим же вхідним масивом чисел, але використовують різні підходи до обробки та підрахунку. Послідовна версія просто обчислює кількість кожного числа послідовно, тоді як паралельна версія розбиває цю роботу на декілька потоків для прискорення процесу.

Отримані часи виконання дозволяють нам зробити наступні висновки:

Послідовний підрахунок зайняв дуже мало часу. Це вказує на те, що виконання послідовного алгоритму є дуже швидким і ефективним на малих обсягах даних.
Паралельний підрахунок зайняв значно більше часу, що означає, що в даному випадку

паралельна версія алгоритму виявилася повільнішою за послідовну. Такий результат може виникнути через надмірні витрати на створення та керування потоками.

Об'єктно-орієнтований дизайн та інкапсуляція:

Функції підрахунку **countNumbersSequential** та **countNumbersParallel** інкапсулюють функціональність залежно від їх призначення. Кожна функція виконує конкретне завдання (послідовний або паралельний підрахунок) і не розголошує свої деталі реалізації на зовнішній рівень.

Дані, такі як вхідний вектор чисел та результати підрахунку, передаються у вигляді параметрів функцій та зберігаються в локальних змінних. Це дозволяє інкапсулювати дані та уникнути глобальних змінних, які можуть збентежити програму.

Код для вимірювання часу виконання розміщений безпосередньо всередині функцій підрахунку. Це дозволяє інкапсулювати логіку вимірювання часу в межах самої функції, що робить код більш чистим та легшим для розуміння.

Висновок:

Послідовний підрахунок:Виконується дуже швидко, займаючи дуже мало часу.Це означає, що використання послідовного підрахунку для малих обсягів даних є доцільним.

Паралельний підрахунок:Займає значно більше часу, ніж послідовний підрахунок.У даному випадку паралельний підрахунок не приніс вигоди, а навіть виявився повільнішим за послідовний, що може бути пов'язано з надмірними витратами на створення та керування потоками.

Об'єктно-орієнтоване програмування та інкапсуляція були суворо дотримані під час реалізації алгоритмів.