**Звіт до Програмного проєкту №2**

***Банасевич Вікторії, варіант №10***

**Експериментальне середовище**

**Кількість апаратних потоків:** 4.

**Компілятор:** Збірка виконувалася у режимах Release (з максимальною оптимізацією) та Debug (без оптимізації).

**Стандарт C++:** C++20.

**Розміри даних:** Дослідження проводилось на векторах з 100 000 , 1 000 000 та 5 000 000 випадкових цілих чисел.

**Операції:**

* **"Швидка операція":** Проста арифметична дія.
* **"Повільна операція":** Обчислювально-інтенсивний цикл для імітації значного навантаження на процесор.

**Аналіз результатів (Режим Release)**

Для аналізу було взято результати для найбільшого набору даних (5 000 000 елементів).

**Порівняння бібліотечних політик виконання**

|  |  |  |  |
| --- | --- | --- | --- |
| **Операція** | **Послідовний (ms)** | **std::execution::par (ms)** | **Прискорення** |
| **Швидка** | 11.570 | 12.426 | 0.93x |
| **Повільна** | 42768.563 | 42460.109 | ~1.01x |

1. Для **"швидкої" операції** приріст продуктивності від паралелізму **відсутній**. Накладні витрати на організацію потоків виявляються більшими, ніж вигода від розпаралелювання простої задачі.
2. Для **"повільної" операції** спостерігається **незначне прискорення** при використанні бібліотечних політик. Це може свідчити про те, що стандартна реалізація std::for\_each з політиками може бути не оптимальною для даного типу навантаження.

**Аналіз власного паралельного алгоритму**

|  |  |
| --- | --- |
| **Кількість потоків (K)** | **Час виконання (ms)** |
| **1** | 42752.347 |
| **2** | 22405.058 |
| **3** | 17597.400 |
| **4** | 14231.252 |
| **5** | 13676.009 |
| **6** | 13565.789 |
| **7** | 13937.104 |
| **8** | 16481.867 |

1. **Найкраща швидкість:** Оптимальна продуктивність була досягнута при K = 6 , що не збігається з кількістю апаратних потоків процесора (4). Це може свідчити про вплив гіпертредингу або особливості планувальника операційної системи, який ефективно розподілив роботу навіть при більшій кількості потоків, ніж фізичних ядер.
2. **Закон зростання часу:** Як видно з таблиці, час виконання спочатку стрімко зменшується зі збільшенням K. Однак після досягнення оптимального значення K=6, час для K=7,8 починає знову зростати. Це явище пояснюється тим, що операційна система витрачає дедалі більше ресурсів на перемикання контексту між потоками. Ці накладні витрати стають більшими за вигоду від паралелізму.

**Вплив оптимізації компілятора**

Для демонстрації важливості оптимізації було проведено запуск програми у двох режимах: Debug (без оптимізації) та Release (з максимальною оптимізацією).

|  |  |
| --- | --- |
| **Режим** | **Час для повільної операції (5 млн ел., 4 потоки)** |
| **Debug** | 29814.619 ms |
| **Release** | 14231.252 ms |

Висновок: Режим Release прискорив виконання програми приблизно у **2.1 рази**.

### **Загальні висновки**

1. Розпаралелювання є ефективним методом прискорення коду, але його доцільність залежить від складності задачі. Найбільший приріст продуктивності досягається для обчислювально-інтенсивних операцій та великих обсягів даних.
2. Власний паралельний алгоритм показав значно краще прискорення для "повільної" операції, ніж стандартні бібліотечні політики.
3. Оптимальна кількість потоків для обчислювальних задач не завжди дорівнює кількості апаратних ядер і може залежати від специфіки задачі та роботи планувальника ОС.
4. Компіляція в режимі Release є критично важливою для отримання реалістичних показників продуктивності.