

## Звіт

Кількість потоків - 8.

Компілятор - G++ (GNU Compiler Collection).

Стандарт C++ - C++20 (-std=c++20).

Розміри даних: Для дослідження використовувалися 3 набори даних розмірами : 10 000, 100 000, 10 000 000 випадкових чисел.

### Порівняння продуктивності стандартної бібліотеки

N (Розмір)	Implicit	std::seq	std::par	std::unseq	std::par_unseq
10 000	0.0105	0.0103	0.0094	0.0096	0.0091
100 000	0.1319	0.1613	0.1119	0.1045	0.1057
10 000 000	16.6780	17.8454	17.3581	18.2107	15.6339

#### Висновки: Найшвидша операція (N=10000000):

Паралельна та векторизована політика std::par\_unseq показала найкращий час (15.6339 мс) для найбільшого обсягу даних.

На великих даних (N=10 000 000) усі паралельні політики (std::par, std::par\_unseq) працюють швидше, ніж послідовні (std::seq), демонструючи вигреш від багатоядерності.

Найповільніша операція (N=10 000 000): Політика std::unseq виявилася найповільнішою серед тестованих для найбільшого розміру масиву (18.2107 мс).

### Порівняння власного паралельного алгоритму

Тестування відбувалося на найбільшому наборі даних  $N = 10\,000\,000$ .

Потоки K	Час (мс)
1	17.4646
2	22.1059
4	26.8362
8	25.5965
16	26.6270

**Висновки:** Версія ( $K=1$ ) виявилася найшвидшою у власній реалізації (17.4646 мс). Виконання з **4 потоками** показало найбільший час (26.8362 мс). Для всіх  $K > 1$  час виконання суттєво зростає, що вказує на неефективність обраної схеми паралелізму.

Зростання часу для  $K > 1$  пояснюється накладними витратами на керування потоками (створення, синхронізація) та, головним чином, домінуючою послідовною фазою об'єднання (`merge_results`), яка обмежує загальне прискорення відповідно до Закону Амдала.

### Загальні висновки

Стандартна бібліотека C++ значно ефективніша у виконанні паралельного інклюзивного сканування завдяки глибокій оптимізації та використанню векторизації. Власна реалізація на `std::thread` виявилася неефективною, оскільки послідовна фаза об'єднання блокує прискорення, роблячи найшвидшою версією ту, що працює в одному потоці.