Звіт з висновками з дослідження

Кількість апаратних потоків (ядер) системи: 16.

Частина 1: Стандартні алгоритми std::min_element

Обсяг (N)	Default/Seq (мкс)	exec::par (мкс)	exec::par_unseq (мкс)	Висновок
10 000 000	20 247	19 194	19 100	Незначний виграш
50 000 000	98 325	97 574	96 135	Незначний виграш

На обох обсягах даних (10M та 50M) паралельні політики (par, par_unseq) не дають значного прискорення порівняно з послідовним виконанням. Усі часи виконання приблизно рівні, різниця становить менше ніж 5%

Операція знаходження мінімуму std::min_element є обмеженою пам'яттю а не обчислювальною потужністю. Продуктивність визначається швидкістю читання даних з основної пам'яті**,** а не швидкістю порівняння. Застосування паралелізму в цьому випадку лише збільшує навантаження на підсистему пам'яті та накладні витрати на керування потоками, що нівелює потенційний виграш.

Власний алгоритм був реалізований з розбиттям роботи на **K** частин та використанням std::async/std::future для паралельного виконання

Обсяг N = 10 000 000$ (мкс)

K	Час (мкс)	K	Час (мкс)
1	20 947	16 (H/W)	4 835 2
10 309	32	4 181	
4	8 312	64 (Best)	3 434
8	5 585		

Найкращий результат**:** K=64, час: 3434 мкс

Співвідношення K_{Best} до апаратних потоків (16): 4.000

Обсяг N = 50 000 000 (мкс)

K	Час (мкс)	K	Час (мкс)
1	101 560	16 (H/W)	16 488
2	51 007	32 (Best)	15 626
4	35 068	64	15 747
8	23 879		

Найкращий результат**:** K=32, час: 15 626 мкс

Співвідношення K_{Best} до апаратних потоків (16): 2.000

Найкраща швидкодія досягається, коли K перевищує кількість апаратних потоків K=16. Це явище є типовим для задач, обмежених пам'яттю.

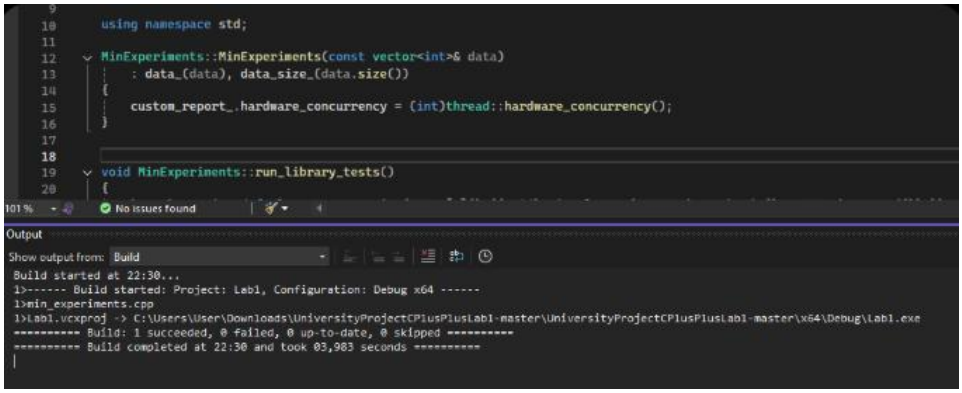
K < H/W (до 16): Час виконання різко зменшується, оскільки задіюється більше фізичних ядер.

K > H/W (після 16): Час виконання, як правило, починає поступово зростати (для N=50M, K=64 повільніше, ніж K=32).Операційна система змушена витрачати час на перемикання контексту між потоками, що перевищують кількість ядер. Це зростання апроксимується лінійним законом, оскільки накладні витрати на додатковий потік стають майже константними.

Для операції std::min_element стандартні паралельні політики C++ неефективні. Однак, власний паралельний алгоритм забезпечує значний виграш у швидкості до 6 разів.

Оптимальна кількість частин K для задач, обмежених пам'яттю, часто перевищує кількість фізичних ядер (16), коливаючись у діапазоні K/H/W від 2.0 до 4.0.

Дослідження продуктивності завжди слід проводити на релізних збірках з увімкненою оптимізацією. Для Memory-bound задач оптимізація компілятора є критичною для мінімізації часу доступу та обробки даних.



```
*** START OF TESTING (N = 10000000 elements) ***
Data generated. First element: 478633

==================================================
EXPERIMENT REPORT (Data size:  10000000)
==================================================

### 1 & 2. Library std::min_element with/without policies ###
+------------------------+-----------------------+
| Algorithm              | Execution time (?s)    |
+------------------------+-----------------------+
| Default/Sequential     |                 31872 |
| std::execution::seq    |                 30784 |
| std::execution::par    |                 30543 |
| std::execution::par_unseq |                 31365 |
+------------------------+-----------------------+
```

```
### 3. Own parallel algorithm (Dependence on K) ###
Number of processor threads (hardware_concurrency): 16
+------+----------------------+
|   K    | Average time (?s)    |
+------+----------------------+
|    1 |               173064 |
|    2 |                84256 |
|    4 |                47717 |
|    8 |                37261 |
|   16 |                24480 |
|   32 |                24147 |
|   64 |                22762 |
+------+----------------------+

### Conclusions regarding our own algorithm ###
- **The best speed** is achieved when **K = 64**, time: 22762 mx.
- Ratio of the best K to hardware_concurrency (16): **4.000**

**The law of increasing time with increasing K:**

*** END OF TESTING (N = 50000000) ***
--------------------------------------------------

C:\Users\User\Downloads\UniversityProjectCPlusPlusLab1-master\UniversityProjectCPlusPlu
process 26504) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debuggi
le when debugging stops.
Press any key to close this window . . .
```