Звіт до програмного проєкту №2

Підготував Цяк Іван, К-24

Спочатку порівняємо результати, отримані у режимах Debug та Release:

Debug

```
-----------------------------------------------
Data size: 100000000

_-_-_-_-_-_-_-_-_-_
Element not found:

Without politics: 268791
Politic std::execution::seq: 228057
Politics std::execution::par: 46604
Politics std::execution::par_unseq: 48131

Speed of own parallel algorithm with different number of threads (K):
K = 1: 266764
K = 2: 652238
K = 3: 566353
K = 4: 414405
K = 5: 401124
K = 6: 333111
K = 7: 376471
K = 8: 360342
K = 9: 386443
K = 10: 399602
K = 11: 401823
K = 12: 432146
K = 13: 431163
K = 14: 423924
K = 15: 419165
K = 16: 435187

Best performance at K = 1 (266764 microseconds)
```

Release

```
-----------------------------------------------
Data size: 100000000

_-_-_-_-_-_-_-_-_-_
Element not found:

Without politics: 26842
Politic std::execution::seq: 27164
Politics std::execution::par: 9730
Politics std::execution::par_unseq: 9542

Speed of own parallel algorithm with different number of threads (K):
K = 1: 40801
K = 2: 27966
K = 3: 18787
K = 4: 14124
K = 5: 11551
K = 6: 12164
K = 7: 15061
K = 8: 13381
K = 9: 12586
K = 10: 12141
K = 11: 11309
K = 12: 10501
K = 13: 11894
K = 14: 12499
K = 15: 13131
K = 16: 14114

Best performance at K = 12 (10501 microseconds)
```

Як бачимо, результати дуже сильно відрізняються. Особливо в Debug версії надзвичайно повільним є паралельний алгоритм. Ці значення є прям неприпустимими для аналізу продуктивності, тому його і треба проводити у режимі Release. Отже, з цього моменту про Debug успішно забуду і буду говорити про результати лиш на Release збірці.

Відкривши файл «output_release.txt», ми можемо побачити, що час виконання виклику std::any_of без політики та з політикою std::execution::seq практично ідентичні (можна сказати ідентичні, бо ще трішки бушує всякий «шум», що спотворює результати, особливо для коротких операцій). Це нам каже про те, що поведінка за замовчуванням є послідовною.

А що ж краще: послідовне чи паралельне виконання? Отже, на малих обсягах даних паралельна політика працювала повільніше за послідовну та за наший — ну, тобто, мій — власний паралельний алгоритм (наш навіть дуже помітно повільний). Але на великих даних паралелізм все ж таки стає кращим і набагато, прям набагато, швидшим.

Проаналізуймо продуктивність власного паралельного алгоритму. Спочатку розглянемо ситуацію, коли елементу не знайдено. Зі збільшенням кількості потоків час виконання спочатку зменшується, досягнувши якогось мінімуму, а потім або зростає, або стабілізується.

Найкраще значення при великих даних досягається при K=12 (хоча через балаган того «шуму» найкраще значення може бути при K дорівнює 10-14), що дорівнює кількості логічних ядер процесора (або інколи є дуже близьким до цього значення). А от далі при більших K час виконання перестає покращуватися, може навіть погіршитись.

От коли шуканий елемент знаходиться в середині, то наш алгоритм працює значно швидше. Але, як бачимо, результати демонструють дуже високу варіативність. От, наприклад, для розміру даних 100000000 при K=2 час склав 295 мс, а от при K=3 — 12726 мс. Ну це просто чиста лотерея, бо все дуже залежить від того, чи потрапив шуканий елемент на початок одного зі шматків даних, чи в кінець. Тому тут при пошуку справжнього елемента дуже все випадково.

Але якщо подивитися загалом, то наш власний алгоритм поступається стандартним паралельним політикам, інколи навіть дуже. Звичайно, подивившись на результати, коли шуканий елемент знаходиться в середині, то при деяких K в нас краще. Але ж це чиста випадковість чи зможемо ми так швидко, чи ні. Отже, краще вже використовувати стандартні паралельні політики, а не винаходити велосипед, бо все ж таки велосипед, винайдений мною, не настільки хороший.