КПІ ім. Ігоря Сікорського

Інститут прикладного системного аналізу

Кафедра Системного проектування

Лабораторна робота №2

З курсу «Паралельні обчисення»

З теми: «Дослідження атомарних змінних та атомарних операцій»

Виконав:

Студент 3-го курсу

групи ДА-21

Сімков Микита

Варіант 20

Київ — 2025

**Мета роботи**

Розглянути поняття атомарності, навчитися працювати з атомарними змінними, а також ознайомитися з підходом написання паралельного коду без блокування.

**Завдання**

1. Ознайомитися з визначенням: атомарна змінна, атомарна операція,

неблокуючий алгоритм. Ознайомитися з деталями атомарності в обраній мові програмування.

2. Надати в протоколі роботи опис того, як саме досягається справжня

атомарність операцій в обраній студентом мові програмування.

3. Виконати завдання за варіантом без використанням паралелізації. Заміряти час виконання завдання.

4. Виконати завдання за варіантом з використанням блокуючих примітивів синхронізації. Заміряти час виконання завдання.

5. Виконати завдання за варіантом з використанням атомарних змінних та CAS\CMPXCHNG операцій (більш високорівнені функції, що

абстрагують дані операції, не приймаються для використання в даній

роботі). Заміряти час виконання завдання.

6. Повторити пункти 2 – 4 з використанням різної розмірності даних та

фіксованою кількістю потоків виконання.

7. В протокол занести отримані для пункту 5 результати у вигляді графіків залежності часу від кількості даних, надати порівняльний аналіз блокуючого та неблокуючого алгоритму з використання атомарних операцій.

8. Надати висновок, що повинен містити аналіз отриманих результатів.

**Хід роботи**

2. В C++ атомарність досягається за допомогою використання низькорівневих інструкцій CAS (Compare-And-Swap). Як зазначено в стандартній бібліотеці <atomic>, атомарні змінні (std::atomic<T>) забезпечують безпечний доступ з кількох потоків без використання блокувань. Основна операція для цього – compare\_exchange\_weak(expected, desired), яка змінює значення, тільки якщо поточне значення дорівнює expected.

3. Мій варіант:

20. Знайти суму трьох максимальних елементів, і самі елементи.

Спочатку реалізація без паралелізації – найзвичайніша реалізація з лінійним проходом по всьому масиву.

Одразу заміряю час:

Для 100000: 0мс

Для 10000000: 7.3мс

Для 1000000000: 778 мс

4. Реалізація з паралелізацією та м’ютексами:

В паралельній реалізації я буду використовувати 6 потоків. Кожен з них буде обробляти 1/6 масиву. Це досягається наступним розподіленням даних: кожен і-тий потік обробляє і-ту комірку та і+6, і +6\*2, і+6\*3 і так до кінця масиву. Що дозволяє потокам не пересікатись. Проте потоки всеодно будуть пересікатись при записуванні в масив, куди я зберігаю знайдені максимуми. Тому там використовуєтсья мутекс, що блокує запис для всіх потоків крім того, який викликав функцію першим.

Заміряний час для 6 потоків (оптимальна кількість для 6 ядер):

Для 100000: 3мс

Для 10000000: 5мс

Для 1000000000: 264мс

5. Реалізація з використанням CAS.

Паралельна реалізація без використання м’ютексів базуєтсья на використанні функції compare\_exchange\_weak, яка виконуєтсья поки не вдасться оновити значення, в іншому ж код аналогічний до звичної паралельної реалізації.

Результати заміру часу:

Для 100000: 4мс

Для 10000000: 5мс

Для 1000000000: 206мс

6-7. Маємо результуючу табличку тестувань:

|  |  |  |  |
| --- | --- | --- | --- |
|  | straight | mutex | CAS |
| 100 000 | 0 | 3 | 4 |
| 10 000 000 | 7 | 6 | 5 |
| 1 000 000 000 | 778 | 264 | 206 |
| 2 000 000 000 | 1669 | 690 | 517 |

A graph with different colored lines

AI-generated content may be incorrect.

Можна побачити, що реалізація з блокуванням та атомарна реалізація мають схожий приріст часу з ростом розміру масиву, проте атомарна реалізація стабільно займає трохи меньше часу.

**Висновок**

В даній лабораторній роботі я ознайомився з атомарними та виконав завдання з використанням CAS та функції compare\_exchange\_weak.

Порівняння:  
Для 100 000 та 10 000 000 код виконуєтсья дуже швидко, тож на створення потоків витрачаєтсья значно більше часу ніж на виконання, тому аналіз для цих розмірів немає сенсу.

Пряма реалізація, як і очікувалось, значно програє в швидкодії та з ростом кількості даних має майже лінійний приріст часу.

Реалізація з використанням м’ютексів для 6 потоків значно покращує швидкодію, проте через потребу в блокуванні все ще не є оптимальним.

Реалізація з використанням атомарних змінних має найкращі показники швидкодії та має стабільну перевагу в часі на 20-25% порівняно з «мутексною реалізацією». Це пов’язано з тим що код виконуєтсья дуже швидко, проте блокування потребує багато часу, тому використання атомарних операцій дозволяє уникнути блокувань, що зменшує накладні витрати на синхронізацію та покращує загальну швидкодію програми.