Отчёт о проделанной домашней работе №3

Была сделана реализация многопоточных счётчиков с помощью 4х способов:

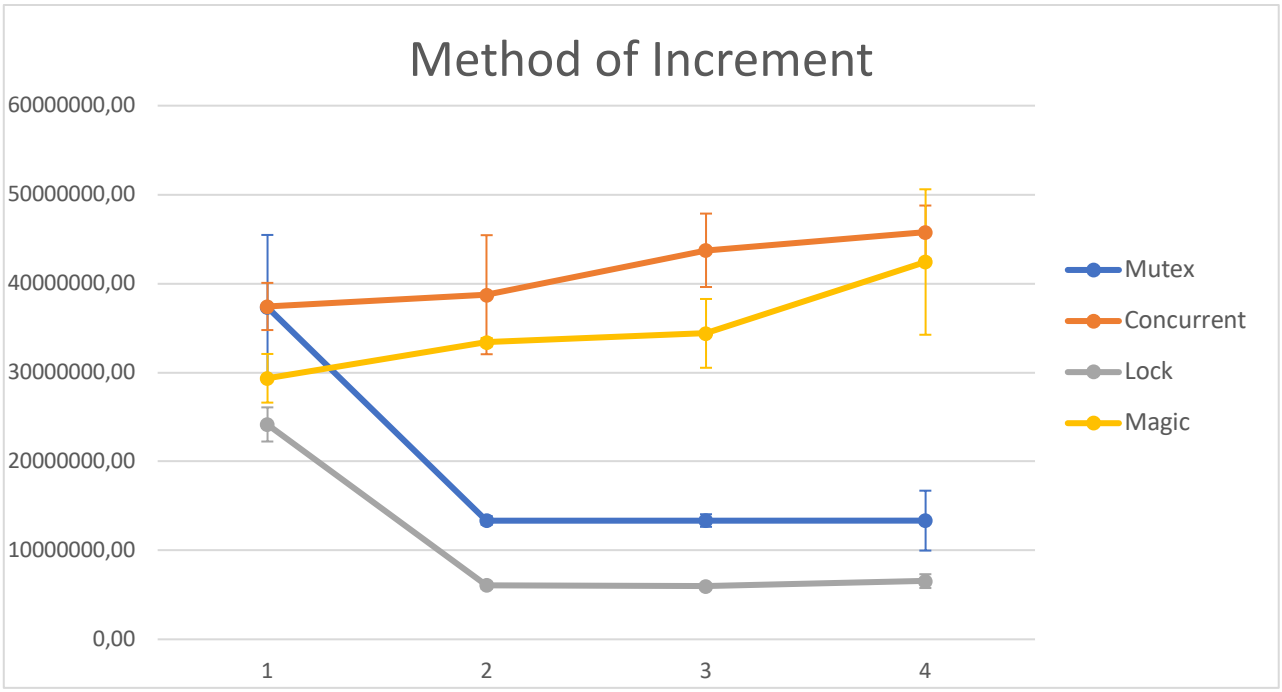
1) С помощью synchronized – блоков (**MutexCounter**)
2) С использованием библиотеки *java.concurrent* (**ConcurrentCounter**)
3) С использованием Locks (**LockCounter**)
4) Без использования примитивов синхронизации (**MagicCounter**)

Реализацию каждого из способов можно увидеть в соответствующих java-классах. Про реализацию MagicCounter-а хочу написать, что для реализации счётчика я использовал HashMap-у. В неё каждый поток пишет и изменяет значение только в своей ячейке. В качестве ключа используется id потока, а значение – счётчик для этого потока. При вызове метода getValue, HashMap сохраняет своё состояние, потом все значения суммируются и передаются.

Также, была измерена производительность каждого из получившихся способов при помощи библиотеки *org.openjdk.jmh.* Реализацию производительности можно увидеть в классе **ForBenchmark**. Результаты бенчмарка можно увидеть в Excel-таблице во вложении (папка *report*). Значения производительности каждого из способов, а в отдельности инкремента и чтения данных предоставлены на соответствующем листе таблицы. Там же можно увидеть и график зависимости производительности от числа потоков.

Далее я хочу прокомментировать конечные значения зависимости производительности от числа потоков:

Производительность для инкремента



По результатам измерения производительности инкремента можно сделать следующие выводы:

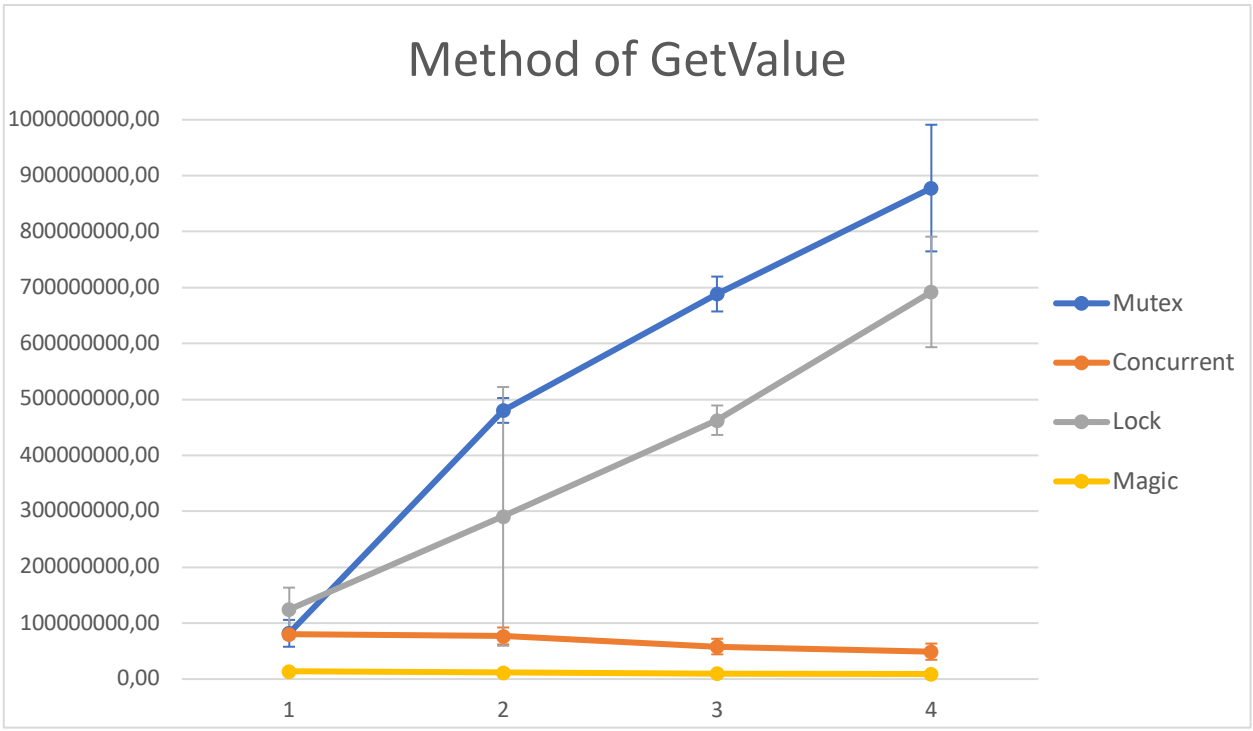
1) Методы, использующие синхронизацию потоков (Mutex & Lock) показывают очень большую производительность при использовании одного потока. С увеличением числа потоков производительность уменьшается. Это происходит потому что потоки ждут, когда освободится монитор. Плюс, примерно одинаковое количество операций в секунду также говорит, что метод инкремента исполняет всегда только один поток.

2) Производительность Lock хуже, чем производительность Mutex.

3) У методов, не использующих синхронизацию потоков (Concurrent & Magic), с увеличением числа потоков увеличивается и число операций в секунду.

4) MagicCounter в производительности не сильно уступает использованию AtomicLong.

Производительность для чтения переменной



Выводы из анализа производительности чтения переменной:

1) Методы Mutex & Lock используют volatile переменную и не используют synchronized – блоки, из чего следует, что при увеличении числа потоков производительность метода getValue растёт.

2) В методах Concurrent & Magic используются более сложные конструкции, поэтому при увеличении числа потоков производительность уменьшается.