Исследование производительности различных реализаций потокобезопасных счетчиков

Информация.

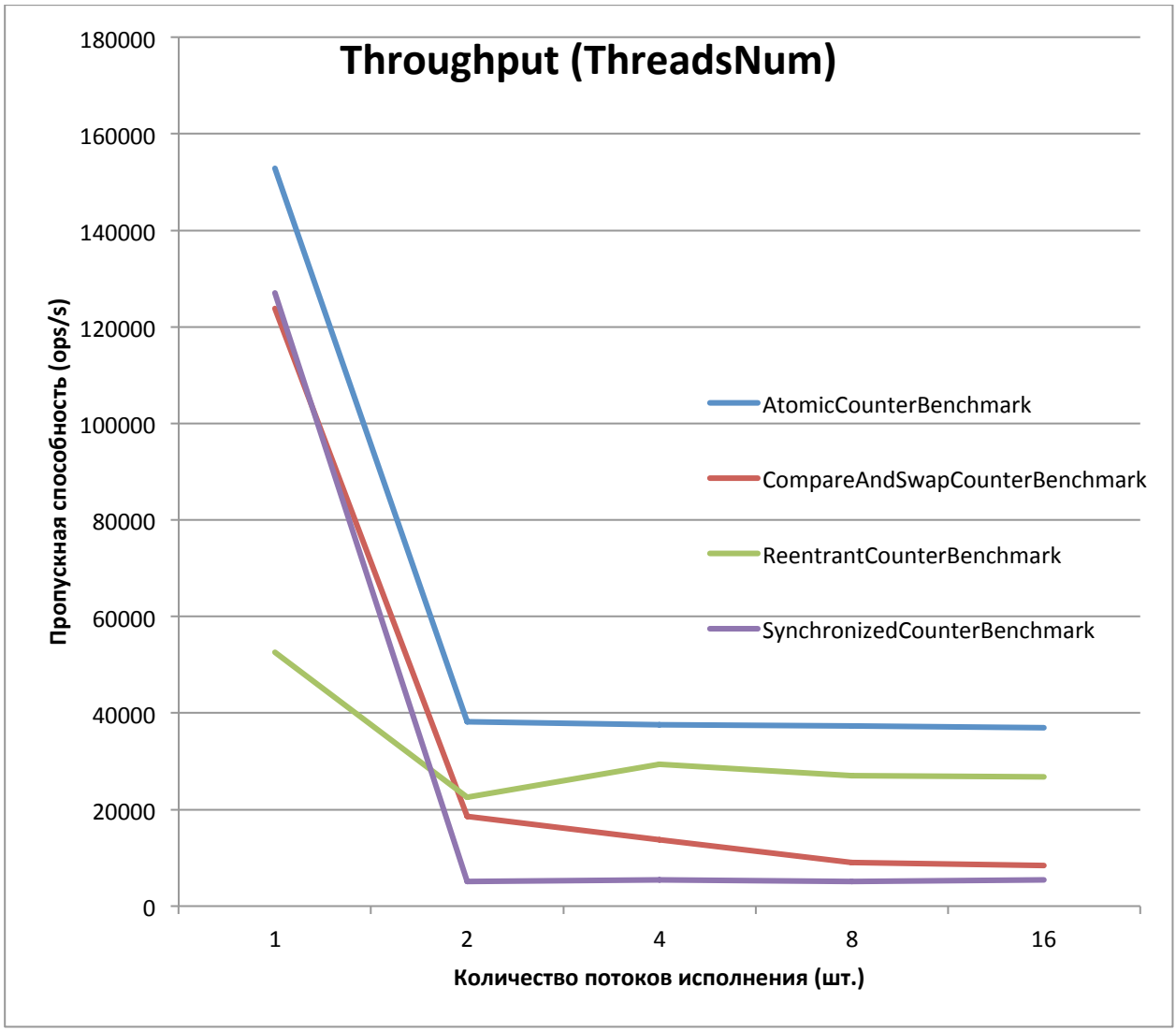
Процессор : Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz

Количество физических ядер: 4

Количество логических ядер: 8 (см. Intel Hyper-threading)

Код: https://github.com/yaroslav11/Multiprocessors/

Результаты измерений:



Дополнительная информация.

Измерения проводились с использованием фреймворка «Java Microbenchmark Harness (JMH)»

Данные усреднены по 5 экспериментальным прогонам, при наличии 1 «прогревочного» прогона.

Выводы.

Счетчики являются конкурентными объектами, на доступ к которым претендует множество потоков. Тем самым создаются накладные расходы, что отрицательно сказывается на производительности.

AtomicCounter

Самый быстрый счетчик. Работает на основе атомарной операции Compare-And-Swap. Эта функция реализована «в железе». Таким образом, по факту получаем аппаратную реализацию неблокирующего алгоритма на спинлоках .

CompareAndSwapCounter

Реализуетт оптимистичный счетчик, который получет значение, увеличивает его, после чего пытается атомарно заменить старое значение на новое, если оно еще не было изменено (на этом шаге используется тот же механизм, что и в AtomicCounter).

Основная идея алгоритма полностью повторяет AtomicCounter, и демонстрирует такую же динамику при увеличении количества потоков. Менее эффективен а счет того, что имеет несколько обращений к объекту, несколько сравнений – то есть больше накладных операций.

ReentrantCounter

Работает на основе ReentrantLock. Блокирующий счетчик. Можно сказать, что он fine-grained, т.к. держит лок только на минимально необходимый кусочек кода.

SynchronizedCounter

При его работе монитор держит блокировку на методе, так что является coarse grained. При этом внутри неявно используется лок, но более сложный, чем ReentrantLock, и с более сложной логикой, чем в случае с ReentrantCounter. Из-за чего данный счетчик и работает медленнее ReentrantCounter, хотя оба они используют локи.