CS131 HW#4
Nicholas Tsianos – 104008515

Results

Due to GetNSet freezing at high numbers of transitions such as 1000000, 100000 transitions was arbitrarily chose. Each state was ran 5 times, and the average is shown in the table below. In addition, 10 threads was arbitrarily chosen, and the following array was used [5 10 15 20 25 30 95 100 105 110].

The correct sum after transitions is 642.

State	**Timing(ns/transition)**	**Sum Mismatch(max difference %)**
Null	2673.53	-
Synchronized	5362.80	-
Unsynchronized	2383.24	40%
GetNSet	3530.68	85%
BetterSafe	3434.89	-
BetterSorry	2617.45	-

Implementations

Unsynchronized

Removed Synchronize keyword from SynchronizedState

GetNSet

Changed Unsynchronized to use AtomicIntegerArray. Constructors updated to convert the array bytes to an array of ints to be passed in to the AtomicIntegerArray Constructor. Swap is composed of 2 simple statements invoking get and set routines.

BetterSafe

Changed GetNSet to use methods getAndDecrement and getAndIncrement instead of a get statement inside of a set statement. Those methods instead function as one atomic operation, and had approximately the same amount of time per transition.

BetterSorry

BetterSorry branches away from AtomicIntegerArrays and was inspired by ReentrantLocks. ReentrantLocks were found to be a nice solution to atomic operations without deadlocks. The constructor creates an array of locks, one for each index of value. Then initiates and tries locks each time swap is called. BetterSorry appears to be the ultimate solution.

Due to the locks, this implementation of BetterSorry is DRF. It does not make sense to implement BetterSorry so that it is worse in some respects than GetNSet and better in others. A possible modification to get rid of DRF is to due a 50/50 system, where if both locks are active, arbitrarily choose a variable to increment or decrement, theoretically achieving a 50% closer sum than GetNSet with the same amount of time per transition.