Homework 3: Java Shared Memory Performance Races

Name
University of California, Los Angeles

Abstract

We want to compare the performance of different synchronization techniques available in Java. This homework illustrates that there is a case where implementing a synchronization without using *synchronized* keyword in Java can speed up the program.

1. Environment

In this section, we will introduce the environments we will be using for the comparison, as well as the benchmark for the testing.

1.1. Java Version

We will use Java version 13.0.2 throughout the test.

1.2. lnxsrv09

One of the machines we will be using is called lnxsrv09, which has 4 CPUs, 8 cpu cores, and 2299.194 cpu MHz.

1.2. lnxsrv10

The other machine we will be using is called lnxsrv10, which has only 1 CPU, 4 cpu cores, and 2095.079 cpu MHz.

2. Testing

Here's the test we will be doing.

1. Create an array of longs
2. Each array entry starts at zero
3. Repeatedly execute a swap operation, which subtracts 1 from one of the entries and adds 1 to an entry (which may or may not be the same entry you subtracted from)

Note the sum of all the array entries should be zero if everything goes well. If the sum is not zero, this indicates some of the transactions (i.e. swaps) didn't go through. Note that it could be the case that multiple transactions failed, but ending up with sum of zero. We can ignore this rare case for now because the chance of this happening is reasonably low. Our goal is to figure out a nice synchronization technique that is fast and complete all the transactions successfully.

2.1. Benchmarks

Let's talk about the test cases to measure the performance of every method. Size of the state array can be 5, 100 or 10000. Number of threads can be 1, 8, 16, or 40. Since we will test on two different machine environments, the combination generates a total of 24 test cases. For each test case, we will perform 100000000 swap operations to make other irrelevant implementation overhead negligible.

2.1. NullState

The swap function of NullState does not do anything. This state shows the fastest synchronization we can achieve because there is no synchronization implemented and yet it returns the sum as 0 (since swap operation is not doing anything and the state array is initialized with 0).

2.1. SynchronizedState

The swap function of SynchronizedState uses the keyword *synchronized* to make sure the swap function is only called one at a time. In other words, when a thread is executing swap operation, it is guaranteed that no other thread is calling swap function.

2.2. UnsynchronizedState

The swap function of UnsynchronizedState does not use the keyword *synchronized* to make sure the swap function is only called one at a time. In other words, when a thread is executing swap operation, another thread could be also calling swap function. This state will show us the importance of synchronization and the cost of not handling race conditions.

2.3. AcmeSafeState

The swap function of AcmeSafeState does not use the keyword *synchronized* to make sure the swap function is only called one at a time. However, instead of using an ordinary long array, it uses AtomicLongArray to represent the state array. AtomicLongArray ensures every change is performed atomically, meaning that it ensures that only one thread can change the state of the array at a time. How is this different from SynchronizedState? Well, SynchronizedState ensures the swap function is performed atomically. In the swap function, we are incrementing a value in the array, and decrementing a value in the array. We don't need to perform these two operations atomically (i.e. we don't need to decrement right after we increment). As long as we decrement it later, it doesn't matter when we do it. For instance, AcmeSafeState allows increment, increment, decrement, and decrement. However, SynchronizedState does not allow this. It only accepts something like increment, decrement, increment, and decrement. Thus, AcmeSafeState has more flexibility updating the state array than SynchronizedState, and thus theoretically runs faster.

3. Analysis

The results of the 24 test cases are shown in the Appendix. Let's analyze the observations we can make from these test results.

3.1. Increasing Number of Threads

Let's talk about how the total time changes as we increase the number of threads. For NullState, the total time decreases as we increase the number of threads. This is happening because we split the work among different threads and since each thread does not do anything, there is no resources to compete for. Thus, the total execution time becomes shorter and shorter. For UnsynchronizedState, there is a very interesting thing happening. As we increase the number of threads, the total time for completion also increases. Then, at one point, the total time starts to decrease. Let's talk about the first part. Here the total time increased probably from the cost of context switching. It didn't happen for NullState because it wasn't doing anything. But we are doing increment and decrement for UnsynchronizedState, and thus there is an opportunity for context switching. Since the total time it took is longer, we can conclude that the cost of context switching outperformed the benefit of splitting the work. Now let's talk about the second part, where the total time starts to decrease again. This is probably when the splitting of the work outperforms the cost of context switching, and thus results in faster total execution time. Let's talk about SynchronizedState. SynchronizedState is pretty much the same as UnsynchronizedState except that it takes longer than UnsynchronizedState because every thread is now competing for the lock to run swap function. However, the benefit is that the operations are synchronized, so the sum is always 0 unlike UnsynchronizedState. AcmeSafeState is an optimized version of SynchronizedState. It performs slightly faster than SynchronizedState, and still gives us the sum of 0 at the end. This is possible because the increment and decrement are performed atomically instead of the entire swap function being atomic.

3.2. Increasing Size of State Array

Let's talk about what happened when we increase the size of the state array. Generally speaking as we increase the size of the state array, the total time it takes doesn't really change (Note you want to compare the ones with the same threads. Also note that the results in Appendix are not very accurate because there were also many different processes other students were running).However, the total time decreases as we increase the size of array for AcmeSafeState. This may not make sense but it actually does! If you look at the implementation of how increment and decrement are done in AcmeSafeState, it uses getAndAdd function, which is defined as follows:

```
public final long getAndAdd(int i, long delta) {
   long offset = checkedByteOffset(i);
   while (true) {
      long current = getRaw(offset);
      if (compareAndSetRaw(offset, current, current +
delta))
         return current;
      }
 }
```

Note compareAndSetRaw is atomic instruction. Thus, when a thread is modifying the state array, it only "locks" the index it's trying to modify instead of the entire state array. Therefore, other threads can still perform its own swap operation if the index it's trying to change is not being used by any other thread. As the size of the state array increases, the number of collisions of the index also decreases, so it results in faster total completion time.

3.3. Using More Powerful Machine

Let's talk about the changes we observe when switching the machine from lnxsrv10 to lnxsrv09. Basically we are increasing the number of cores and thus allowing more threads to be run concurrently without/less competing for resources. What do I mean? Well, if you have only a few cores, then when I run say 100 threads, then since there is not enough cores, it will waste its time by spending most of the time context switching rather than doing the actual work. But if I have more cores, then it can handle more threads. So if you look at the results in the Appendix, you see when you run at different servers under the same conditions, lnxsrv09 (the one with more powerful machine) generally performs better. For instance, consider that we are using 40 threads. Suppose machine A has 8 cores, and machine B has 2 cores. Then each core in machine A would run 5 threads, and each core in machine B would run 20 threads. A thread in machine B must compete for resources because there are 19 other threads running on the same core. But machine A only has 5 threads per core. Thus, machine A would give us better performance. Therefore, it makes sense that lnxsrv09 is performing better than lnxsrv10.

Appendix

lnxsrv09	state array size 5
1 thread	Total time 1.33882 s real, 1.33752 s CPU Average swap time 13.3882 ns real, 13.3752 ns CPU
8 threads	Total time 0.270987 s real, 2.11801 s CPU Average swap time 21.6790 ns real, 21.1801 ns CPU
16 threads	Total time 0.288857 s real, 3.98980 s CPU Average swap time 46.2172 ns real, 39.8980 ns CPU
40 threads	Total time 0.441380 s real, 10.3262 s CPU Average swap time 176.552 ns real, 103.262 ns CPU
lnxsrv09	state array size 100
1 thread	Total time 1.40459 s real, 1.39626 s CPU Average swap time 14.0459 ns real, 13.9626 ns CPU
8 threads	Total time 0.295125 s real, 2.25865 s CPU Average swap time 23.6100 ns real, 22.5865 ns CPU
16 threads	Total time 0.314781 s real, 4.51147 s CPU Average swap time 50.3649 ns real, 45.1147 ns CPU
40 threads	Total time 0.323580 s real, 7.99664 s CPU Average swap time 129.432 ns real, 79.9664 ns CPU
lnxsrv09	state array size 10000
1 thread	Total time 1.33011 s real, 1.32881 s CPU Average swap time 13.3011 ns real, 13.2881 ns CPU
8 threads	Total time 0.282421 s real, 2.22867 s CPU Average swap time 22.5937 ns real, 22.2867 ns CPU
16 threads	Total time 0.306163 s real, 4.26618 s CPU Average swap time 48.9860 ns real, 42.6618 ns CPU
40 threads	Total time 0.307600 s real, 6.39575 s CPU Average swap time 123.040 ns real, 63.9575 ns CPU
lnxsrv10	state array size 5
1 thread	Total time 1.14699 s real, 1.14585 s CPU Average swap time 11.4699 ns real, 11.4585 ns CPU
8 threads	Total time 0.617923 s real, 1.49578 s CPU Average swap time 49.4339 ns real, 14.9578 ns CPU
16 threads	Total time 0.444331 s real, 1.22923 s CPU Average swap time 71.0929 ns real, 12.2923 ns CPU
40 threads	Total time 0.500577 s real, 1.23955 s CPU Average swap time 200.231 ns real, 12.3955 ns CPU
lnxsrv10	state array size 100
1 thread	Total time 1.07474 s real, 1.06983 s CPU Average swap time 10.7474 ns real, 10.6983 ns CPU
8 threads	Total time 0.457556 s real, 1.22869 s CPU Average swap time 36.6045 ns real, 12.2869 ns CPU
16 threads	Total time 0.602973 s real, 1.34773 s CPU Average swap time 96.4756 ns real, 13.4773 ns CPU
40 threads	Total time 0.575196 s real, 1.33931 s CPU Average swap time 230.078 ns real, 13.3931 ns CPU
lnxsrv10	state array size 10000
1 thread	Total time 1.07359 s real, 1.06816 s CPU Average swap time 10.7359 ns real, 10.6816 ns CPU
8 threads	Total time 0.525387 s real, 1.32842 s CPU Average swap time 42.0310 ns real, 13.2842 ns CPU
16 threads	Total time 0.420448 s real, 1.27291 s CPU Average swap time 67.2716 ns real, 12.7291 ns CPU
40 threads	Total time 0.666216 s real, 1.74080 s CPU Average swap time 266.486 ns real, 17.4080 ns CPU

NullState

lnxsrv09	state array size 5
1 thread	Total time 2.32158 s real, 2.24521 s CPU Average swap time 23.2158 ns real, 22.4521 ns CPU
8 threads	Total time 25.2277 s real, 85.6137 s CPU Average swap time 2018.22 ns real, 856.137 ns CPU
16 threads	Total time 29.9906 s real, 103.621 s CPU Average swap time 4798.49 ns real, 1036.21 ns CPU
40 threads	Total time 25.6656 s real, 87.3699 s CPU Average swap time 10266.2 ns real, 873.699 ns CPU
lnxsrv09	state array size 100
1 thread	Total time 2.08730 s real, 2.08623 s CPU Average swap time 20.8730 ns real, 20.8623 ns CPU
8 threads	Total time 27.9847 s real, 88.7606 s CPU Average swap time 2238.78 ns real, 887.606 ns CPU
16 threads	Total time 27.4235 s real, 86.8381 s CPU Average swap time 4387.76 ns real, 868.381 ns CPU
40 threads	Total time 27.6215 s real, 86.3843 s CPU Average swap time 11048.6 ns real, 863.843 ns CPU
lnxsrv09	state array size 10000
1 thread	Total time 2.44118 s real, 2.43809 s CPU Average swap time 24.4118 ns real, 24.3809 ns CPU

8 threads	Total time 24.9135 s real, 71.1550 s CPU Average swap time 1993.08 ns real, 711.550 ns CPU
16 threads	Total time 10.5772 s real, 22.4469 s CPU Average swap time 1692.35 ns real, 224.469 ns CPU
40 threads	Total time 11.3953 s real, 24.5526 s CPU Average swap time 4558.14 ns real, 245.526 ns CPU
lnxsrv10	state array size 5
1 thread	Total time 1.78527 s real, 1.77775 s CPU Average swap time 17.8527 ns real, 17.7775 ns CPU
8 threads	Total time 5.19714 s real, 6.39654 s CPU Average swap time 415.771 ns real, 63.9654 ns CPU
16 threads	Total time 5.45549 s real, 6.81454 s CPU Average swap time 872.878 ns real, 68.1454 ns CPU
40 threads	Total time 5.44215 s real, 7.54731 s CPU Average swap time 2176.86 ns real, 75.4731 ns CPU
lnxsrv10	state array size 100
1 thread	Total time 1.70084 s real, 1.69793 s CPU Average swap time 17.0084 ns real, 16.9793 ns CPU
8 threads	Total time 4.50575 s real, 4.73932 s CPU Average swap time 360.460 ns real, 47.3932 ns CPU
16 threads	Total time 4.46003 s real, 4.84869 s CPU Average swap time 713.605 ns real, 48.4869 ns CPU
40 threads	Total time 4.89907 s real, 6.16635 s CPU Average swap time 1959.63 ns real, 61.6635 ns CPU
lnxsrv10	state array size 10000
1 thread	Total time 1.75798 s real, 1.75594 s CPU Average swap time 17.5798 ns real, 17.5594 ns CPU
8 threads	Total time 4.03011 s real, 4.39319 s CPU Average swap time 322.409 ns real, 43.9319 ns CPU
16 threads	Total time 3.91185 s real, 4.10909 s CPU Average swap time 625.896 ns real, 41.0909 ns CPU
40 threads	Total time 4.41860 s real, 5.04105 s CPU Average swap time 1767.44 ns real, 50.4105 ns CPU

SynchronizedState

lnxsrv09	state array size 5
1 thread	Total time 1.49302 s real, 1.49189 s CPU Average swap time 14.9302 ns real, 14.9189 ns CPU
8 threads	Total time 4.88060 s real, 38.5976 s CPU Average swap time 390.448 ns real, 385.976 ns CPU output sum mismatch (-24652 != 0)
16 threads	Total time 2.58329 s real, 40.0055 s CPU Average swap time 413.327 ns real, 400.055 ns CPU output sum mismatch (10920 != 0)
40 threads	Total time 3.29298 s real, 88.8430 s CPU Average swap time 1317.19 ns real, 888.430 ns CPU output sum mismatch (-7714 != 0)
lnxsrv09	state array size 100
1 thread	Total time 1.49159 s real, 1.49035 s CPU Average swap time 14.9159 ns real, 14.9035 ns CPU
8 threads	Total time 4.56831 s real, 36.2147 s CPU Average swap time 365.465 ns real, 362.147 ns CPU output sum mismatch (-17043 != 0)
16 threads	Total time 3.58052 s real, 56.1159 s CPU Average swap time 572.882 ns real, 561.159 ns CPU output sum mismatch (-43970 != 0)
40 threads	Total time 3.15353 s real, 83.0279 s CPU Average swap time 1261.41 ns real, 830.279 ns CPU output sum mismatch (-25185 != 0)
lnxsrv09	state array size 10000
1 thread	Total time 1.50885 s real, 1.50759 s CPU Average swap time 15.0885 ns real, 15.0759 ns CPU
8 threads	Total time 1.85670 s real, 14.1860 s CPU Average swap time 148.536 ns real, 141.860 ns CPU output sum mismatch (-10219 != 0)
16 threads	Total time 1.93362 s real, 30.5365 s CPU Average swap time 309.380 ns real, 305.365 ns CPU output sum mismatch (-11329 != 0)
40 threads	Total time 1.99084 s real, 51.1442 s CPU Average swap time 796.336 ns real, 511.442 ns CPU output sum mismatch (-14337 != 0)
lnxsrv10	state array size 5
1 thread	Total time 1.21387 s real, 1.20882 s CPU Average swap time 12.1387 ns real, 12.0882 ns CPU
8 threads	Total time 3.35916 s real, 10.1497 s CPU Average swap time 268.733 ns real, 101.497 ns CPU output sum mismatch (-7060 != 0)
16 threads	Total time 2.19606 s real, 7.65482 s CPU Average swap time 351.370 ns real, 76.5482 ns CPU output sum mismatch (-894 != 0)
40 threads	Total time 1.93059 s real, 6.42598 s CPU Average swap time 772.236 ns real, 64.2598 ns CPU output sum mismatch (-3536 != 0)
lnxsrv10	state array size 100

1 thread	Total time 1.22832 s real, 1.22464 s CPU Average swap time 12.2832 ns real, 12.2464 ns CPU
8 threads	Total time 3.54511 s real, 10.5647 s CPU Average swap time 283.609 ns real, 105.647 ns CPU output sum mismatch (-14988 != 0)
16 threads	Total time 3.51519 s real, 11.5830 s CPU Average swap time 562.430 ns real, 115.830 ns CPU output sum mismatch (-12175 != 0)
40 threads	Total time 4.22052 s real, 15.2804 s CPU Average swap time 1688.21 ns real, 152.804 ns CPU output sum mismatch (-16606 != 0)
lnxsrv10	state array size 10000
1 thread	Total time 1.25096 s real, 1.24984 s CPU Average swap time 12.5096 ns real, 12.4984 ns CPU
8 threads	Total time 2.04072 s real, 6.02312 s CPU Average swap time 163.258 ns real, 60.2312 ns CPU output sum mismatch (-4710 != 0)
16 threads	Total time 2.02651 s real, 6.61815 s CPU Average swap time 324.242 ns real, 66.1815 ns CPU output sum mismatch (-6328 != 0)
40 threads	Total time 2.03501 s real, 6.53829 s CPU Average swap time 814.004 ns real, 65.3829 ns CPU output sum mismatch (-4710 != 0)

UnsynchronizedState

lnxsrv09	state array size 5
1 thread	Total time 2.66879 s real, 2.66735 s CPU Average swap time 26.6879 ns real, 26.6735 ns CPU
8 threads	Total time 13.8909 s real, 106.302 s CPU Average swap time 1111.27 ns real, 1063.02 ns CPU
16 threads	Total time 11.7175 s real, 184.885 s CPU Average swap time 1874.79 ns real, 1848.85 ns CPU
40 threads	Total time 11.0020 s real, 312.524 s CPU Average swap time 4400.81 ns real, 3125.24 ns CPU
lnxsrv09	state array size 100
1 thread	Total time 2.68392 s real, 2.68271 s CPU Average swap time 26.8392 ns real, 26.8271 ns CPU
8 threads	Total time 10.4340 s real, 83.1909 s CPU Average swap time 834.722 ns real, 831.909 ns CPU
16 threads	Total time 6.31407 s real, 99.7652 s CPU Average swap time 1010.25 ns real, 997.652 ns CPU
40 threads	Total time 4.43716 s real, 123.633 s CPU Average swap time 1774.86 ns real, 1236.33 ns CPU
lnxsrv09	state array size 10000
1 thread	Total time 2.89633 s real, 2.87772 s CPU Average swap time 28.9633 ns real, 28.7772 ns CPU
8 threads	Total time 1.91095 s real, 15.1198 s CPU Average swap time 152.876 ns real, 151.198 ns CPU
16 threads	Total time 1.27386 s real, 19.4063 s CPU Average swap time 203.818 ns real, 194.063 ns CPU
40 threads	Total time 1.36984 s real, 32.6561 s CPU Average swap time 547.937 ns real, 326.561 ns CPU
lnxsrv10	state array size 5
1 thread	Total time 2.54325 s real, 2.53350 s CPU Average swap time 25.4325 ns real, 25.3350 ns CPU
8 threads	Total time 12.4763 s real, 39.3084 s CPU Average swap time 998.105 ns real, 393.084 ns CPU
16 threads	Total time 14.5992 s real, 51.3592 s CPU Average swap time 2335.87 ns real, 513.592 ns CPU
40 threads	Total time 5.41224 s real, 19.4084 s CPU Average swap time 2164.89 ns real, 194.084 ns CPU
lnxsrv10	state array size 100
1 thread	Total time 2.52213 s real, 2.51145 s CPU Average swap time 25.2213 ns real, 25.1145 ns CPU
8 threads	Total time 5.26059 s real, 15.5349 s CPU Average swap time 420.847 ns real, 155.349 ns CPU
16 threads	Total time 4.56135 s real, 15.4735 s CPU Average swap time 729.815 ns real, 154.735 ns CPU
40 threads	Total time 5.94919 s real, 21.9216 s CPU Average swap time 2379.67 ns real, 219.216 ns CPU
lnxsrv10	state array size 10000
1 thread	Total time 2.72353 s real, 2.72076 s CPU Average swap time 27.2353 ns real, 27.2076 ns CPU
8 threads	Total time 2.81377 s real, 8.40259 s CPU Average swap time 225.101 ns real, 84.0259 ns CPU
16 threads	Total time 2.95237 s real, 9.11571 s CPU Average swap time 472.380 ns real, 91.1571 ns CPU
40 threads	Total time 2.56809 s real, 8.66045 s CPU Average swap time 1027.23 ns real, 86.6045 ns CPU

AcmeSafeState