Brian Rodriguez
U0853593

Quiz 3:

2. In VGood.java the use of volatiles ensures that the threads will act asynchronously, mean that the other thread running the code in the else statement will know when the volatiles change. In the VBad.java the threads never know when the variables change, so one of the threads can be hung up in a while loop indefinitely. On my machine the hang-up happened ranging from 6-8 iterations.

3. Volatile declarations signals a synchronization requirement for the compiler. When multiple threads run the same code with the shared variable, they work in a synchronized patter and that when the variable is changed the other thread knows about the change. Java Byte-code does not have any notable traces of the volatile variable being initialized. The machine code however, does have a a signal of locking when decompiled.

4. The author of the Memory barriers JVM concurrency talks about the memory barriers and fences produced when using java. The author mentions that it is crucial that the compiler follows the unidirectional and bidirectional locking mechanism. If the compiler does not follow these rules, then there is something wrong in the code.

5. The article demonstrates the smoothness of being able to print the binary code generated by the program you have created.

6. The memory fence is realized through extraneous testing of reads and writes involving volatile variables. In the graphs, it is easy to see that the extraneous tests really shine through and aid in the realization of memory fences. It depicts that the processors must make the operations visible to other threads. This overhead slows down the program but a large margin, but more work is also done in the process.

7. The volatiles and atomic variables are significantly more expensive than using unsynchronized variables. These synchronized variables cost about 100x for writes and about 25x for reads across both Java and C++11. This is because the program has to signal to the other threads a synchronization section has been hit, along with other under the hood operations that must be done.

9. In the VGood.java program provided, it is easy to see that with volatiles the program will not have deadlocks. It follows the 4-phase handshake process put into place. When a thread has the id of 0, the next thread will be put into the else statement. The else statement contains the necessary operations to keep the other thread going to through the loop. The volatiles will not change until one thread is finished handling the operations of the acquired volatile in both cases.

10. Reading the code in sequential semantics, it is possible to see that if the code was actually compiled in this order then there would be no deadlocks. When talking about thread creations, each thread created will theoretically be executing the code it was given before other threads are created. This ensures that the program order is kept intact. For this example of VGood.java, the first thread that is ran will set req equal to true, so that the second thread can continue and set ack equal to true. Then sequentially req will be set to false then ack set to false. This is promised by sequentially reading the program and by the rules of happens-before.

11. Loop interchanging is the process of switch the order in which the loops execute. This is done to improve spatial and temporal cache locality to increase the performance of the program.

12. Section 5.2.2 gives an in-depth analysis of Translation-Lookaside Buffers. TLBs can be a very powerful tool when utilized correctly but can also be a hindrance as the translation of the TLB to main memory fetches can cost several cycles; by this I mean that translating from a virtual memory space to main memory can cost several cycles of computation.

13. A loop unroll is the technique of doing i+x number of computations per iteration of a for-loop. For example, if a loop is executing result += i, then in loop unrolling we would do result += i+1… i+2 in the same iteration of that loop. Then increment i+=3 on the next iteration of the loop. After a certain threshold this does not pay dividends and can actually hinder the performance of the program.

14. Outer loop unrolling is essentially the same process for loop unrolling except that the inner loops do j + x in every iteration of the outer loop. This does more work per iteration but can also hinder performance and give diminishing returns if a certain threshold is hit.

15. Unroll and jam is a concept of taking two inner for loops and packing them together in the same loop while the outer for loop is using the unroll technique to spread the workout over an iteration. The two inner for loops then do the same work but in less time.

16. The loop interchange is illegal because the memory access of the two arrays is different. Since i and j are being switched the ordering and operations of accesses will also be switched yielding different results.

17. Loop fusion is the process of merging two smaller loops into a larger loop. This improves the reads in the cache to be used more frequently as it does not have to load in more pages from memory.

18. Loop fission is the technique of transforming a loop into several loops. This technique is used to improve the performance of a single loop to prevent more fetches to main memory and utilize the cache.

19. Loop tiling or blocking is a powerful technique to utilize spatial and temporal cache locality. It breaks the loops into chunks and accesses fewer pages of memory per outer loop iteration. In the example of matrix transposition, it is possible to see that two loops will overlap at a certain point thus using the same cache locations and prevent fetching from main memory.

20. The halide language is a programming language created for synthesizing the optimization of image processing. It enables simpler programming of high-performance image processing code. The author explains that the loop optimizations are very powerful but are completely dependent on the architecture that the program is being executed on. In image processing it is spatially and computationally heavy. The amount of memory needed to make these optimizations efficient requires powerful architecture.