Parallel Programming

Part A

1. **What is a race condition?**
   1. Race condition is a computer’s behavior where a result output is controlled by unexpected events. This causes a bug in the program that needs to be fixed
2. **Why race condition is difficult to reproduce and debug?**
   1. There are two distinct conditions for a race condition. First, one being it is nondeterministic. Second, the program depends on relative timing among the several threads. The problems caused, don’t show up through the debugging mode, so the programmer is required to go through the program step-by-step.
3. **How can it be fixed? Provide an example from your Project A3?**
   1. The threads are synchronized from one thread to another. Synchronizing these threads makes one thread visible to another, this is similar to sharing the data among threads. An example from A3 could be using “Uniform Memory Access (UMA)” through OpenMP, this allows the memory to be shared among the threads. Through this we can task a thread by “check and act” via another thread.
4. **Summarize the Parallel Programming Patterns section in the “Introduction to Parallel Computing\_3.pdf” in your own words?**
   1. To perform a repetitive task effectively the computer uses patterns. Since a computer does repetitive tasks a lot, it has many patterns and as a programmer one should most importantly consider the strategies on those patterns, such as Concurrent Execution Mechanisms Start. The programmer to make his program run effectively and use the computer in the fastest way must know the algorithms that helps him achieve that. A Concurrent Execution Mechanism though is a strategy it is used by and related to the computer architecture and not so much to programming. But a programmer must always know what it does and how it functions along with the processors he uses and the threads he computes and important techniques such as parallel programming (OpenMP). For Parallel Programming Patters there are two key coordination patterns: 1. message passing; 2. mutual exclusion.
5. **In the section “Categorizing Patterns” in the “Introduction to parallel computing\_3.pdf” compare the following: collective synchronization (barrier) w/ collective communication; master-worker with fork join.**
   1. Collective synchronization vs Collective communication: these are both a type of categories in Concurrent Execution Mechanisms. In the collective synchronization a barrier system is used, while in the collective communication a reduction system is used. In the barrier system used by collective synchronization threads from all the groups are synchronized together. This system also blocks all systems until all the group members answer the call. Whereas, in the collective communication the program is returned to all group members and only verify the result to one member.
   2. Master-worker vs Fork join: these are both a type of pattern categorized as Program Structure. In a fork join parallelism, a program gets merged and the data at one single point so that the sequential execution is resumed. This is fork (work separately initially) join (then merge into one). Whereas in the master-worker parallelism pattern program is run simultaneously across multiple threads
6. **Where can we find parallelism in programming?**
   1. The program view is where parallelism can be found. In parallelism the program is executed line-by-line and any line/lines that do tasks simultaneously represent the parallel program.
7. **What is dependency and what are its types?**
   1. Dependency is a type of program whose operations require the previous operation’s result so that it can compute the current operation. There are three types of dependencies: flow dependency, output dependency, anti-dependency. Ex. of flow dependency – r = 5, a = r. Ex. of output dependency – r = 7, previous r = 6. Ex. of anti-dependency – r = s, s = 9.
8. **When is a statement is dependent and when is it independent?**
   1. Dependent when the previous statement’s output affects the output of the current statement, ex. r = 3, s = r, s is dependent upon the output of r. Independent when the previous statement does not affect the output of the current statement, ex. r = 3, s = 7, both s and r not relying on each other to compute their result.
9. **When can two statements be executed in parallel?**
   1. Two statements can be executed parallelly when the processors run simultaneously.
10. **How can dependency be removed?**
    1. Dependency can be removed by simply rearranging the statements and by eliminating statements.
11. **How do we compute dependency for the following two loops and what type of dependency?**

**for(i = 0; i < 100; i++) for(i = 0; i < 100; i++){**

**S1: a[i] = i; S1: a[i] = i;**

**S2: b[i] = 2\*i;**

**}**

* 1. The first statement on both sides is dependent on i. This is an example of flow dependency since a is constantly changing based on i. While the second statement on the right is an example of anti-dependent since statement 1 is performed first and then statement 2 is but statement 1 does not affect statement 2.