**Task 3: Parallel Programming Skills**

1. Foundation

* Race condition:

1. What is race condition?

* A race condition is the behavior of an electronics, software, or other system where the output is dependent on the sequence or timing of other uncontrollable events. It becomes a bug when events do not happen in the order the programmer intended.

1. Why race condition is difficult to reproduce and debug?

* Race condition is difficult to reproduce and debug since the end result is nondeterministic and depends on the relative timing between interfering threads. Problems occurring in production systems can disappear when running in debug mode, when additional logging is added, or when attaching a debugger.

1. How can it be fixed? Provide an example from your Project\_A2 (see spmd2.c)

* Processes or threads should not depend on some shared state. Operations upon shared states are critical sections that must be mutually exclusive. Obeying this rule will not open up the possibility of corrupting the shared state. It is better to avoid race conditions with careful software design rather than attempting to fix them afterwards.
* In spmd2.c file from Project A2, variables id and numThreads are declared outside of the block that will be forked and run in parallel. This results in all threads sharing that variable’s memory location. This is a problem with the code, and it needs to be fixed by declaring the variables inside the block and making each thread to keep track of its id separately.
* Summarize the Parallel Programming Patterns section in the “Introduction to Parallel Computing\_3.pdf” (two pages) in your own words (one paragraph, no more than 150 words).
* Parallel programs contain many useful ways of writing code that have been used and documented by developers over time. These patterns can be grouped into two main categories: strategies and concurrent execution mechanisms. In regard to strategies, programmers should consider algorithmic strategies and implementation strategies for parallel programming. First, algorithmic strategies choose what tasks are being done concurrently. Second, implementation strategies construct the overall program structure, including how the data being computed is structured. On the other hand, concurrent execution mechanisms primarily consist of two major patterns. First, process/thread control patterns dictate how the processing units are controlled at run time. Second, coordination patterns set up how tasks that are running concurrently coordinate to finish the computation desired (message passing between concurrent processes or mutual exclusion between concurrent threads). In addition, a hybrid computation implementing both of these patterns is another emerging type of parallel implementation.
* In the section “Categorizing Patterns” in the “Introduction to Parallel Computing\_3.pdf” compare the following:

1. Collective synchronization (barrier) with Collective communication (reduction)

* Collective synchronization manages the sequence of work and the tasks performing. Each task performs its work until it reaches the barrier and stops. All tasks are synchronized when the last task reaches the barrier.
* Collective communication involves a group of processes and affects a data transfer between them. By reduction, it combines the data from all processes, returning the result to the root process.

1. Master-worker with fork-join

* In master-worker structure, a master process sets up a group of worker processes and a task queue. Then the workers execute concurrently, with each worker removing a task from the queue and processing it. In fork-join structure, a main process forks off a number of other processes and continues in parallel to accomplish the work. Then those processes rejoin the main process at the end.
* Dependency: Using your own words and explanation, answer the following:

1. Where can we find parallelism in programming?

* In the program, parallelism can be found in between program statements by focusing on which statements can be executed concurrently. It can also be found in larger-grained statements in block level, loop level, routine level, or process level. We can find parallelism by examining data and resource, such as how the data is operated on or where the data resides.

1. What is dependency and what are its types (provide one example for each)?

* A dependency arises when an operation depends on a previous operation and result.
* True dependence occurs when an instruction depends on the result of a previous instruction (ex. S1: a = 1; S2: b = a).
* Output dependence occurs when the ordering of instructions affects the final output value of a variable (ex. S1: a = f(x); S2: a = b).
* Anti-dependence occurs when an instruction requires a value that is later updated (ex. S1: a = b; S2: b = 1).

1. When a statement is dependent and when it is independent (Provide two examples)?

* If the order of execution does not matter, the statements are independent of each other (ex. S1: a = 1; S2: b = 1). If the order of the execution affects the computation, the statements are dependent (ex. S1: a = 1; S2: b = a).

1. When can two statements be executed in parallel?

* Two statements can be executed in parallel when there are no dependences between them.

1. How can dependency be removed?

* Some dependences can be removed my rearranging or eliminating statements.

1. How do we compute dependency for the following two loops and what type/s of dependency?



* To compute dependency for the loops, we need to unroll them into separate statements and show dependences between those statements.
* These two loops do not contain dependences between the iterations.