**Parallel Programming Skills**

* (15p) Race condition:
  + (2p) What is race condition?

**Also known as race hazard, is an electronics’, software’s or other system’s behavior where the result is depends on sequence or timing g uncontrollable events. This can cause issues when event do not happen in the correct order. This usually occurs in logic circuits, and in multithreaded/distributed programs.**

* + (5p) Why race condition is difficult to reproduce and debug?

**Race conditions can be difficult to reproduce and debug, because the final result can not be determined as it depends on the relative timing between interfering threads. Problems that occur in the production system(s) can disappear in debug mode when more logging is added or when attaching a debugger (mostly referred to as “Heisenbug”). Because of this, programmers usually avoid race conditions by carefully designing the program rather than going back and fixing them.**

* + (8p) How can it be fixed? Provide an example from your Project\_A2 (see spmd2.c).

**To fix race condition errors, we need to declare variable in a way that each forked thread has their own private variable instead of a shared variable. This is because critical race conditions usually occur when processes depend on shared state. Operations that uses a shared state are important sections that must be mutually exclusive. Looking back at the spmd2.c program from project A2, if we initialized id and numThreads outside of the #pragma omp parallel block, we will get race conditions. This is because the forked threads will all share the same id and numthreads. To fix this issue, we simply initialize id and numThreads inside the #pragma omp parallel block, which allows each forked thread to have its own private id and numThreads.**

* (15p) Summaries 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).

**In the strategies pattern, software developers have to decide what algorithmic strategies to use and given that, the implementation strategies to use. The algorithmic strategy deals with what tasks can be done concurrently by multiple processors working concurrently. Implementation strategy deals with the overall structure of the program or its data structure. In concurrent execution pattern, there are two major categories: process/thread control and coordination pattern. Process control pattern dictates how processes units are controlled during runtime (we have been using this pattern in the previous two projects). Coordination pattern sets up how multiple concurrently running tasks coordinate to complete the computation. There are two major coordination patterns: massage passing and mutual exclusion. Massage passing uses Message Passing Interface (MPI) to pass messages between concurrent processors on multicore machines(s). Mutual exclusion uses OpenMP to share memory applications between threads running on a single shared memory system. Another type of parallel implementation uses hybrid computation, where it uses bother of the coordination patterns on a cluster of machines.**

* (12p) In the section “Categorizing Patterns” in the “Introduction to Parallel Computing\_3.pdf” compare the following:
  + Collective synchronization (barrier) with Collective communication (reduction)
    - **Collective Synchronization**
      * **how multiple processes will join up at a certain point**
      * **how multiple copies of a dataset are kept in coherence with one another to maintain data integrity**
    - **Collective Communication**
      * **Each process performs the same communication operations**
      * **Communication and computation are corresponding to group of processes in a communicator**
  + Master-worker with fork join
    - **Master-worker**
      * **Master is responsible of assigning and receiving tasks to and from the worker, and workers are responsible for executing certain sub-task**
      * **After worker completes the task, the result is then submitted to the master, where it will be summarized**
    - **Fork join**
      * **Execution branches off (fork) in parallel at certain point in the program execution, and then merges (join) at a following point to resume sequential execution**
      * **Programs fork (make new) threads when needed, and when a thread finished execution, it will join with the program**
* (26p) Dependency: Using your own words and explanation, answer the following:
  + (3p) Where can we find parallelism in programming?
    - **Statement Level**
      * **Between program statements**
    - **Block/loop/routine/process level**
      * **Larger-grained program statements**
  + (6p) What is dependency and what are its types (provide one example for each)?

**Dependency is when one operation depends on an earlier operation to finish before it can be executed. There are two types of dependency (Fundamental [concurrent] execution assumption and sequential consistency). In fundamental execution assumption, processors run independently from each other and no assumptions are made about their processing speed. For instance, if we want to execute two statements in parallel, we can have processor 1 execute statement 1 while processor 2 executes statement 2. In sequential consistency, the statements that are executed do not interfere with each other, and they produce the same computation results. For example, if we have two statements to execute in parallel, there are two possibilities on how these statements will execute. In the first case, statement 1 can be executed by processor 1 first while statement 2 waits on statement 1 to finish processing before being processed on processor 2. In the second case, statement 2 can be executed by processor 2 first while statement 1 waits on statement 2 to finish processing before being processed on processor 1.**

* + (3p) When a statement is dependent and when it is independent (Provide two examples)?

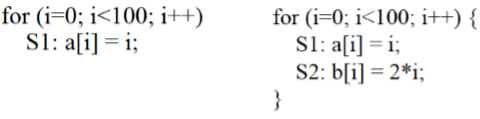
**A statement is independent when their order of execution does not matter. For example, setting variable A to equal something, then setting variable B to equal something else. A statement is dependent when their execution order affects the computation result. For example, setting variable A to equal something, then setting variable B to equal variable A.**

* + (3p) When can two statements be executed in parallel?

**Two statements can be executed in parallel if and only if there are no dependency between the two statements.**

* + (3p) How can dependency be removed?

**Dependence can be removed by modifying the program, like rearranging or eliminating statements.**

* + (8p) How do we compute dependency for the following two loops and what type/s of dependency?
    - **For the first for loop (true dependence):**

* + - **For the second for loop (true dependence):**