Name: Minh Binh Nguyen

PantherID: 002-46-4288

**Project A4 - Task 3 Report**

**Part a: Foundation**

**Race condition:**

**What is race condition?**

A race condition is a bug of an electronics, software, or other system where the output is depended on the timing of other uncontrollable events. It can occur especially in logic circuits, in multithreaded or distributed software.

**Why race condition is difficult to reproduce and debug?**

Race condition is known to be difficult to reproduce and debug because the final output is hard to be determined, and it heavily depends on the relative timing between interfering threads.

**How can it be fixed? Provide an example from your Project\_A3(see spmd2.c)**

We can fix it by declaring the variables inside the pragma part, so that each thread can have its own private copy of the variables. In the spmd2.c example, we declare the variable id inside the pragma part, that’s why each id is unique for each thread.

**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).**

Parallel Programming contains many patterns which can be grouped into two main categories: Strategies and Concurrent Execution Mechanisms. With strategies, the programmer should be asking themselves two questions: what algorithmic strategies and what implementation strategies to use. Parallel algorithmic strategies are the choices to be made about what tasks can be executed concurrently using multiple cores processing units. Parallel programs can use several patterns of implementation strategies. Some of those affect the overall structure of the program, and others care about how the data can be computed by multiple processing units. The concurrent execution patterns can be grouped into two major categories: Process/Thread control patterns (which control at runtime how the processing units run), and Coordination patterns (which help multiple concurrently running tasks on processing units coordinate to complete the parallel computation desired). Most software uses one of the two coordination patterns: message passing between concurrent processes, and mutual exclusion between threads executing concurrently.

**In the section “Categorizing Patterns” in the “Introduction to Parallel Computing\_3.pdf” compare the following:**

**Collective synchronization (barrier) with Collective communication (reduction)**

Barrier is a point where all the threads must stop and cannot proceed until all other threads reach this point.

Reduction lets you specify thread-private variables that are subject to a reduction operation at the end of the parallel region.

**Master-worker with fork join**

Master-worker: The work will be divided. The thread with id 0, called master, executes one block of code, the rest, called the workers, executes another block of code.

Fork-join: The fork assigns the child thread with some piece of work, the child thread performs the work, then call the join to exit.

**Dependency: Using your own words and explanation, answer the following:**

**Where can we find parallelism in programming?**

In programming point of view, we can find parallelism in the statement level (we can decide which statement can be executed at the same time), or in the block level / loop level / routine level/ process level (we can decide which block of code/ which loop/ which routine/ which process can be executed at the same time.

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

A dependency is where an operation has to wait for a previous operation to complete, in order to use its output to perform. There are 3 types of dependency: true dependence, output dependence, and anti-dependence.

Examples:

True dependence:

S1: a = 1;

S2: b = a; // S2 depends on S1

Output dependence:

S1: a = f(x);

S2: a = b; // S2 depends on S1

Anti-dependence:

S1: a = b; // S1 depends on S2

S2: b = 1;

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

A statement is considered to be independent when its output is not affected by the other statements.

A statement is considered to be dependent when its output is affected by the other statement(s).

Example:

int a = 3;

int b = 7;

These statements are independent.

int a = 3;

int b = a + 4;

These second statement is dependent. It depends on the first statement.

**When can two statements be executed in parallel?**

Two statements can be executed in parallel when they are not dependent. In other word, they don’t have any type of dependencies (true dependencies, anti-dependencies, output dependencies).

**How can dependency be removed?**

A dependency can be removed by rearranging or eliminating statements. In other word, it can be avoided by having good data architecture, remodeling the code and algorithm.

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

**A screenshot of a cell phone

Description automatically generated**

Loop 1: S1 depends on the variable i in the loop statement. This is True dependency.

Loop 2: S1 and S2 depends on the variable i in the loop statement. This is True dependency. Notice that S1 and S2 don’t depend on each other.

**Part b: Programming**

I started with getting my Raspberry Pi connected to my computer and typed in the code for the trap-notworing program (Figure 1).

**A screenshot of a cell phone

Description automatically generated**

Figure 1

The result of the program should be 2.0, but I ended up getting random numbers, not 2.0 as expected (Figure 2).

A screenshot of a cell phone

Description automatically generated

Figure 2

Then I typed in the working version as instructed (Figure 3).

A screenshot of a cell phone

Description automatically generated

Figure 3

This version works as it’s supposed to, we get 2.0 no matter how many threads we give it. We achieve this because we used reduction operator for the integral variable (Figure 4).

A screenshot of a cell phone

Description automatically generated

Figure 4

Then I moved on to the next part by typing in the code for barrier program (Figure 5).

A screenshot of a social media post

Description automatically generated

Figure 5

Before uncommenting the statement, we got the pairs of “before – after” for each thread. This is because the threads try to finish their whole job when they’re called (Figure 6).

A screenshot of a cell phone

Description automatically generated

Figure 6

After uncommenting the statement as instructed, the output is organized in the pattern of all the “before” statements are performed first by the threads. All threads wait for each other to finish the “before” part first, then they move on together to the “after” part (Figure 7).

A screenshot of a cell phone

Description automatically generated

Figure 7

Moving to the next part, I typed in the code as instructed (Figure 8).

A screenshot of a social media post

Description automatically generated

Figure 8

This is what we get before uncommenting the parallel statement (Figure 9).

A screenshot of a social media post

Description automatically generated

Figure 9

Without the parallel statement, there is only 1 thread be generated, and it always has its ID as 0, which is the master.

After uncommenting the parallel statement, the program now produce the outputs with the number of threads that we want (Figure 10).

A screenshot of a cell phone

Description automatically generated

Figure 10

In this parallel pattern, the first thread executes a block of code (the thread with ID 0, also known as the master), and the rest of the threads execute the other block (the workers).