CSC3210 Project 4 Task 3 – Tony Ngo

**Task 3A**

1. **What is race condition?** 
   1. Race Conditions happen when one system’s behavior is dependent on the timing of other uncontrollable events.
2. **Why race condition is difficult to reproduce and debug?** 
   1. Since timing is random, the race condition can disappear while debugging, which is also referenced as a “Heisenbug”
3. **How can it be fixed? Provide an example from your Project\_A3 (see spmd2.c)**
   1. An example of how this could be fixed is creating instances outside of the portion that initializes the use of OpenMP Pragmas. This is because when you initialize the instances while utilizing the threads, there is room for randomness because we are not sure which threads will be initialized, thus creating a “Heisenbug”.
4. **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).**
   1. In coding, there are two types of patterns: strategies and concurrent execution mechanisms. There are two primary types of strategies: algorithmic and implementation strategies given the algorithmic strategies. Algorithmic Strategies make choices about which thread to execute concurrently (aka the most efficient way to execute a program), while the implementation strategies contribute to certain processes or the overarching processing unit.

Concurrent Execution Mechanisms are parallel code patterns that enable parallelism or concurrent execution- an example would be OpenMP Pragmas which utilizes multithreading. There are two different concurrent execution patterns: Process/Thread control or Coordination Patterns. Process/Thread control patterns control how many threads/what threads are executed at a time, while coordination patterns set up how many running tasks are needed to complete the pattern. Coordination patterns set up how many threads/processes are going to be setup.

There is a third emerging implementation which utilizes both MPI and OpenMP in one program (hybrid computation), which can run multiple computers in a cluster.

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

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

**o Master-worker with fork join**

* 1. Masterworker takes use of multiple processors through the line of code “#pragmas OpenMP parallel” because we are able to use threading. Collective synchronization is used in barrier because we can see both cases where parallel execution is utilized (before we removed the comments on barrier) and then when synchronized execution is utilizes (after we removed the comments on barrier). Both instances of code are similar in the way that they utilize the OpenMP library to accomplish the threading. However, in Barrier (Collective Synchronization), it utilize both sequential and parallel patterns, while Masterworker only utilizes sequential.

1. **Where can we find parallelism in programming?**
   1. Parallelism is found wherever multithreading is found, so in any video editing application or program that utilizes high level computations.
2. **What is dependency and what are its types (provide one example for each)?**
   1. A dependency is when an operation depends on an earlier operation to produce the result. One type is execution two statements in parallel and another is sequential consistency in parallel execution.
      1. Two statements in parallel

A screenshot of a cell phone

Description automatically generated

* + 1. Sequential Consistency in Parallel Execution

A screenshot of a cell phone

Description automatically generated

1. **When a statement is dependent and when it is independent (Provide two examples)?**
   1. A statement is independent if the order of execution does not matter. A statement is dependent if the order of the execution will affect the outcome.
      1. Independent Execution

A close up of a clock

Description automatically generated

* + 1. Dependent Execution

A close up of a sign

Description automatically generated

1. **When can two statements be executed in parallel?**
   1. Statements can be executed in parallel when the statement order does not matter and has dependencies.
2. **How can dependency be removed?**
   1. Dependencies can be removed if they are independent of one another.
3. **How do we compute dependency for the following two loops and what type/s of dependency?**
   1. Loop-Level Parallelism
      1. Dependency is calculated within loops
   2. Iteration Space
      1. Undo loop and separate statements/iterations & show dependences between iterations

**Task 3B**

A screenshot of a cell phone

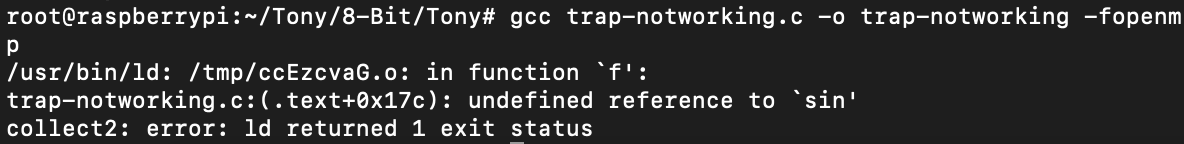
Description automatically generated

This is the code for trap-notworking.c shown using the nano text editor.

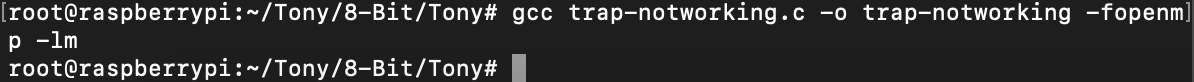
A screenshot of a cell phone

Description automatically generated

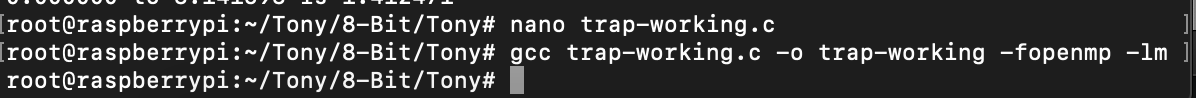
Here is the code for trap-working.c; I also used the nano GNU to write this.



When I tried to compile it using “gcc trap-notworking.c -o trap-notworking -fopenmp”, it would not let me compile because it couldn’t recognize the “sin(x)” portion of my code.



After doing some online research, I was able to find that adding “-lm” fixed it because it is a part of the “math” package.



Here is the compilation for “trap-working.c” using “gcc trap-working.c -o trap-working -fopenmp -lm”, I also wrote it using the nano GNU.

A screen shot of a social media post

Description automatically generated

Here is the output for trap-notworking and trap-working. As we can see the correct output was displayed for trap-working. However, trap-notworking is giving an incorrect answer, this is because on line 37 in trap-working, they initialized the value in 37/38 on a separate line, instead of like on line 37 on trap-notworking which initializes it on one line.

A screenshot of a cell phone

Description automatically generated

This is what the barrier code looks like with the omp barrier still commented out; this code was written with the nano GNU. I had to re-write the code because copy and paste would mess up the format of the code.

A screenshot of a cell phone

Description automatically generated

I was able to compile the code easily both times. When you uncomment the omp barrier, it makes the threads run in order instead of parallel. This is because when you uncomment the barrier it is not running in parallel anymore, the threads are now executing in sequential order.

A screenshot of a cell phone

Description automatically generated

This is what the masterWorker.c code looks like with the pragma OpenMP parallel commented out; this code was written using the nano GNU. I also had to write the code instead of copying it over because the code because the formatting was messed up.

A screenshot of a cell phone

Description automatically generated

This is the compilation using “gcc masterWorker.c -o masterWorker -fopenmp” of masterWorker before and after the uncommenting of the pragma OpenMP parallel. Before we uncommented it and ran the code, there was only one thread running at a time, after we uncommented and ran the code, there was four threads running at a time. This is because when the code is commented, we did not utilize threading, but after we uncomment it, the code utilizes all threads.