VEDANSI PARSANA

TASK 3A)

* Race condition:
* What is race condition?

🡪 A race condition (race hazard) it’s the condition of an electronics, software, or other system where the system's functional behavior is dependent on the sequence or timing of other uncontrollable events. When programmer wants to do any event and it doesn’t happen than it becomes a bug.

* Why race condition is difficult to reproduce and debug?

🡪 Race condition is difficult to reproduce and debug because debugging race conditions in C/C++ Bugs that are hard to reproduce pull up time and energy of any software development team. Therefore, one cause of these bugs can be race conditions, which causes unpredictable behavior and make getting a reliable bug report which is nearly impossible. The difficulty in locating the race conditions is because nothing really goes wrong with the program unless a trigger is activated to do so.

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

🡪 In order to avoid the race condition from happening is to plan and write the code design for the program.

Here is an example from the spmd2.c: one of the thread’s id is appearing more than once.

When compiled and ran, the program printed out something similar to this:

Hello from thread 1 of 4

Hello from thread 2 of 4

Hello from thread 1 of 4

Hello from thread 2 of 4

To fix this issue, we commented out the initial declaration in line 5 and declared id and numThreads as the int data type.

A screenshot of a cell phone

Description automatically generated

* Summaries the Parallel Programming Patterns section in the “Introduction to Parallel Computing\_3.pdf” (two pages) in your own words

🡪 There are to main topics discussed in the parallel programming patters that is strategies and concurrent execution mechanism. Strategies apparatuses parallel programming and further divides into two categories Implementation and Algorithmic. Implementation determines how are the tasks of program are being processed. Algorithmic determines how and when tasks should be executed. Concurrent Execution Mechanism, which relates to the hardware where different operations executes concurrently. The Concurrent Execution Mechanism are divided into two major subcategories: Process/Thread control patterns and Coordination patterns. The Process/Thread control patterns controls which thread to process and when will this happen at the runtime. Third implementation involves hybrid computation with combination of Strategies and Concurrent Execution Mechanisms. This pattern uses both OpenMP and Message Passing Interface.

* In the section “Categorizing Patterns” in the “Introduction to Parallel Computing\_3.pdf” compare the following: Collective synchronization (barrier) with Collective communication (reduction) and Master-worker with fork join.

🡪 In **Collective communication**, all the processes are reach to a specific point before executing. It acts as a reduction as once process of the communicator collects data from all other processes and performs an operation to find the result. It uses MPI\_Reduce () function. In **Collective Synchronization**, it blocks all the processes until a specific synchronization point is being reached. It acts as a barrier since it blocks the process until all other processes are being reached to synchronization point successfully. It uses MPI\_Barrier () function. In **master-worker** pattern, a main process is being divided into small pieces which in turn being distributed to several worker processes whereas in **Fork-Join** pattern which is used to execute parallel light weight processes and threads.

* Dependency: Using your own words and explanation, answer the following:
* Where can we find parallelism in programming?

🡪  Parallelism is a mechanism that enables programs to run faster by performing several computations at the same time by incorporating hardware with multiple CPU's. mobiles, databases, servers, virtual reality and man more.

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

🡪 Dependency itself tells that one element is depending on other elements. basically, all our programs will have this dependency in-built. in case of DBMS, we will have dependencies like trivial, non-trivial, multivalues, transitive etc.

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

🡪 A statement is dependent if its values is getting computed using previously assigned or computed values.

example: a=10 b=20 c = a + b here statement c = a + b is dependent on a=10 and b=20 statements

A statement is not dependent if its values are independent of previous values.

example: a=10 b=20 c=45+65-90 printf ("hello") here print statement and c=45+65-90 statements are independent of other statements i.e. a=10 and b=20

* When can two statements be executed in parallel?

🡪 two statements can execute in parallel if they don’t share any common data item that is getting updated in two statements.

* How can dependency be removed?

🡪 By making multiple statements i.e. for example by applying Normalizations like 1nf, 2nf, 3nf, bcnf, 4nf, 5nf in dbms which creates multiple tables with less data.

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

🡪Here in the leftmost for loop, in instruction S1: a[i] = i, the dependency exist is

a[i] is dependent on variable i and since a[i] value will be written only after reading variable i, hence a[i] is dependent on variable i and the dependency type is RAW which is also called Real dependency. Hence the dependency is a[i] dependent on i with RAW I.e. Real dependency.

In the rightmost for loop, in instruction S1: a[i]= i, a[i] is dependent on variable i with RAW I.e. Real dependency and in instruction S2: b[i] = 2\*i, b[i] is dependent on i with RAW.

RAW = Read After Write.

Hence dependencies are:

1. a[i] dependent on i with RAW.

2. b[i] dependent on i with RAW.

Task 3B)

The first part of parallel programming task four, I created both files and executables for trap-notworking and trap-working. Then after their creation I created the executables for each one, however due to the use of the sin method, I had to include -lm at the end of the gcc creation command.

A screenshot of a cell phone

Description automatically generated

From the output of the programs I can see that trap-notworking fails to receive the correct number, the difference between #pragma omp parallel for with \ and without, is that using it will save the last iterations data for use outside the #pragmas… clause.

Barrier:

The barrier class had to tests to run, using it without the included #pragma omp barrier and using it with it.

(without using the barrier)

A screenshot of a cell phone

Description automatically generated

(using barrier)

A screenshot of a cell phone

Description automatically generated

The threads without the barrier printed out there before and after statements in the order they finished. When running with the barrier however, all the threads stopped once they got to the barrier in the code until all other threads caught up.

In the workerMaster program, I created a program and ran it with and without the use of #pragma omp parallel.

(workerMaster without pragma)

A screenshot of a cell phone

Description automatically generated

(workerMaster witH pragma)

A screenshot of a computer

Description automatically generated

After creating the excutable and running the code, I saw that without the use of parallel, the code would of course only output one line, the master, as the code never splits or forks into separate threads so only thread 0, the master, runs. Once I use the parallel code output the separate threads are created thus allowing for the worker threads to also reply.