**Parallel Programming Skills: Foundation**

**(15p) Race condition:**

**(2p) What is race condition?**

The behavior of your program when the output is dependent on the sequence or timing of uncontrollable events.

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

The end result is not able to be determined and depends on relative timing between the interfering threads, so it can change every time you run it

**(8p) How can it be fixed? Provide an example from your Project\_A3 (see spmd2.c)**

It can be fixed by careful software design. Here is where an example occurs in Project A3

int parallelSum(int\* a, int n) {

int sum = 0;

int i;

#pragma omp parallel for reduction(+:sum)

for (i = 0; i < n; i++) {

sum += a[i];

}

return sum;

}

Without the reduction statement, the threads might try to write to sum at the same time, causing a data race or race condition. The solution is “careful software design”, that is the reduction statement. This statement gives each thread its own variable copy so they do not interfere with each other, then all get combined back into the original one after they are done.

**- (15p) Summaries the Parallel Programming Patterns section in the**

**“Introduction\_to\_Parallel\_Computing\_4.pdf” (two pages) in your own words (one**

**paragraph, no more than 150 words).**

There are two main categories developers use for writing parallel programs: Strategies and concurrent execution mechanisms. You should consider an algorithmic strategy that is concerned with making choices about what tasks can be done concurrently by multiple units. Based on your algorithmic strategy consider what implementation strategy to use; these are about the overall structure of the program and how the data being computed is structured. There are two main categories of concurrent execution mechanisms: Process/thread control patterns(how processing units are controlled at runtime), and coordination patterns(how multiple running tasks coordinate)

**- (12p) In the section “Categorizing Patterns” in the**

**“Introduction\_to\_Parallel\_Computing\_4.” compare the following:**

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

Collective synchronization is when the threads must all be completed before any can move on to the next step. There is a barrier preventing them from continuing in this case. Collective communication is when all threads communicate in some way to come to a result. Reduction makes individual copies of the variable in the reduction clause and gives one to each thread. This process implies it makes a barrier as well. Before the variable in the reduction clause can be used again, the threads must all finish their individual actions on their private variable and be joined together again.

**o Master-worker with fork join**

Pragma forks threads and joins them back together. Master-worker is when during the forking, you make statements assigning a particular thread to do a task(the master), and the other threads do other tasks(The workers)

**- (26p) Dependency: Using your own words and explanation, answer the following:**

**(3p) Where can we find parallelism in programming?**

We can find parallelism in between program statements and when statements can be done at the same time. Within loops or other program statements, there can be parallelism for the statements in them.

**(6p) What is dependency and what are its types (provide one example for each)?**

Dependency is where operations cannot execute without the execution of another operation. The types of dependencies are control, data, and system.

Control: An if loop like if(i < 0) {i++}. The execution of the statement i++ is control dependent on the condition that i is less than 0.

Data: An example would be the following two statements:

A = 1;

B = A;

Here B is dependent on the data of A

System: An example would be when there are multiple computers in a system at a company. There could be a task that requires the use of multiple computers, and thus there is system dependency between them

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

**examples)?**

Statements are independent when the order of the statements does not matter and dependent when the order does matter.

Dependent example:

DATE\_BORN = 1997

AGE = CURRENT\_DATE – DATE\_BORN

The AGE variable is dependent on DATE\_BORN being defined first. Without this statement, AGE could not be calculated

Independent example:

AGE = 22

NAME = “NICK”

Here the AGE and NAME variables are completely unrelated. No matter the order the result will be the same.

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

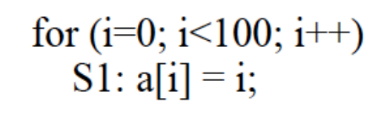
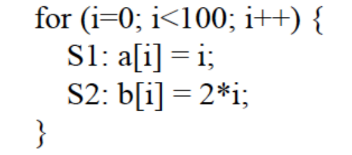
Two statements can be executed in parallel if there is no dependencies between them.

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

You can remove dependency by modifying the program in ways like rearranging statements and eliminating statements.

**(8p) How do we compute dependency for the following two loops and what type/s of**

**dependency?**

**a) b)**

You unroll the loop into separate iterations and show the dependencies between the iterations.

a)Each iteration is only affecting a unique index of the array a, and thus they have no dependency between each other. The only dependency here is that S1 will affect a different index based on what i is.

b)The two statements here are independent of each other, because no calculation in either are used in the other one. They could be reversed and the same result would occur. In addition to this, the statements are independent of the statements in the next iteration for the same reason as the explanation for a). Like with the previous loop, the only dependency is the statements on what i is each iteration.