HW8.5. Fun with Parallelism

Feel free to check out the guide that we have prepared to help you in this problem.

In a multithreaded system, any order of operations could possibly happen, from each thread happenening serially, to all of them running at the exact same time. For example, if you have four threads, all 4 threads will execute (and inevitably finish) that same code flow. However, the threads do not necessarily start at the same time; the threads do not necessarily execute at the same rate (the code that each thread executes is not perfectly interleaved); the threads do not necessarily finish running the whole program at the same time.

For these sets of problems, let's explore what happens when you try to execute the same code on multiple threads.

Q1: We run the following code on four threads:

```
#pragma omp parallel
{
// Hint: loop variable i is declared inside the #pragma directive
// therefore, it is a private variable per thread.
    for (int i = 0; i < 3; i++) {
        printf("%d",i);
    }
}
```

Q1: Select all of the following outputs that are possible:
Hint: Think of the expected output of a single thread first, then imagine what happens if more threads start executing their own loops independently with different levels of interleaving.

- ☐ (a) 012012012012
- ☐ (b) 010012201212
- ☐ (c) 010011201212
- ☐ (d) 100001222112
- ☐ (e) 000011112222
- ☐ (f) 000111102222

Select all possible options that apply. ?

Q2: Say we have the following RISC-V code to be run by multiple threads:

```
li t0 10
sw t0 0(s0)
lw t0 0(s0)
add t0 t0 t0
sw t0 0(s0)
```

Hint: Note that the memory location being accessed by each thread is the same (i.e. it is a shared variable). Think about the access pattern that would result to different threads reading a different data from the shared memory.

Q2.1: What is the **smallest** possible value stored at `0(s0)` after two threads finish executing the code?

integer ?

Q2.2: What is the **largest** possible value stored at `0(s0)` after two threads finish executing the code?

integer ?

Q2.3: Now, let's extend it further. What is the **largest** possible value stored at `0(s0)` after four threads finish executing the code?
Hint: Answer Q2.2 correctly first to understand the execution pattern needed to get the worst case scenario.

Homework 8

Assessment overview

Total points: 0/100

Score: 0%

Question

Value: 40

History:

Awarded points: 0/40

Report an error in this question

Previous question

Next question

Attached files

No attached files

Attach a file

Attach text

integer

Q3: We run the following code on two threads:

```
// Hint: The following variables are declared outside the #pragma directive
// therefore, these variables are shared for all threads.
int y = 0;
int x = 3;
#pragma omp parallel
{
    while(x > 0) {
        y = y + 1;
        x = x - 1;
    }
}
```

Note that the expression `y = y + 1` is equivalent to three instructions: load value of `y`, add 1, store result to `y`. Another thread can execute in between those instructions. Moreover, `y` is a shared variable, which means that any changes that one thread does to `y` will be seen by the other thread. The same can be said for variable `x`. Exploit this fact when answering the questions below. It might be helpful if you can answer Q2 first before answering this one since you can imagine breaking down the C code into the assembly for a more fine-grained analysis.

Q3.1: What is the **smallest** value `y` can contain after the code runs?
Hint: One thread can complete the entire loop while the other thread has just started.

integer

Q3.2: What is the **largest** value `y` can contain after the code runs?
Hint: Since `x` is relatively small (loop count is small), try writing out the different `y` values for each thread as they execute. If you want to maximize the value of `y`, how can two threads 'force' more iterations to happen?

If `x` was initialized to 2 instead of 3, the largest possible value for `y` would be 5. Think about how to get this.

integer

Q3.3: Now, let's extend it further. If `x` was initialized to **10** instead of 3 (still a shared variable), what is the **largest** value `y` can contain after the code runs?
Hint: Answer Q3.2 correctly first to understand the execution pattern needed to get the worst case scenario.

If you answered Q3.2 by brute force (by writing all cases as each thread executes), you can find a formula on the worst case value of `y` given the initial value of `x` to help you answer this question.

integer

Save & Grade *20 attempts left* Save only *Additional attempts available with new variants*