Wes Chouza

CS 4522 Midterm

All answers to discussion questions must be in complete sentences and grammatically correct!

1. Explain what is meant by cache coherence and why it is so critical in parallel programming.

2. What is meant by "thread safe"? In general, how would you modify code that is not "thread safe" in order to make it "thread safe"?

3. Pthreads supports busy-waiting, mutexes, and semaphores as tools to implement mutual exclusion. Discuss the strengths and weaknesses of each.

4. There are limits on the amount of speed-up that can be achieved using parallel programming techniques. Explain why this is true.

5. What is loop dependence (loop carried dependence, data dependence)? Why is it important in openmp?

6. Write a pthreads program (C/C++-style pseudocode) to calculate the dot product of 2 vectors of floats. Make your "best guess" for the names and arguments of the pthread functions. Do not write include statements, error handling code, or code that gets the data. Start from the place that you have valid data.

Wes Groover

100/100

1. Cache coherence is the idea that while each thread/process may have a local copy of a variable in it's own cache, the integrity of that variable is protected everywhere. In other words, if a copy of a variable is updated in one cache, every other place that variable is stored needs to be updated as well. This is important in parallel programming because race conditions can give erroneous results that the developer must ensure against.

14/14

2. Thread-safe means that one thread's access of a variable does not affect future access of any other thread. Code that is not thread safe can be made so by using local variables where possible and updates to globals are protected in a critical section.

14/14

3. Busy-wait: Pros: Easy to implement, efficient when critical section execution faster than context-switch overhead.

Cons: Can dramatically degrade performance when many threads waste CPU cycles in while-loops.

14/14

Mutex: Pros: Efficient use of resources compared to busy-wait. Much faster with many threads

Cons: No control of order of access. Incorrect implementation can result in deadlock.

Semaphore: Pros: Efficient use of resources compared to busy wait. Can allow multiple access to resource. Ordered access.

Cons: Most difficult and dangerous to implement

4. BECAUSE ANY PORTION OF A PROGRAM THAT IS NOT PARALLELIZED WILL NOT BE SPED UP. FOR EXAMPLE, IF A PROGRAM IN SERIAL TAKES 20 SECONDS AND 50% OF THE PROGRAM IS CONVERTED TO PARALLEL, THE SERIAL PORTION WILL STILL TAKE 10 SECONDS NO MATTER HOW MANY PROCESSORS YOU HAVE

14/14

5. LOOP DEPENDENCE IS WHEN DATA IN ONE PART OF A LOOP IS DEPENDANT ON DATA IN ANOTHER, SUCH AS COMPUTING THE FIBONACCI SEQUENCE. THIS IS IMPORTANT IN OPENMP BECAUSE OF THE PARALLEL FOR STRUCTURE. THE RESPONSIBILITY FOR GUARDING AGAINST LOOP DEPENDANCE FALLS ON THE DEVELOPER

14/14

6.
```
VOID* LOCAL_DOT(VOID* RANK);
FLOAT GLOBAL_DOT = 0;
LONG THREAD_COUNT;
LONG VECTOR_LENGTH;
MAIN(ARGS){
  FLOAT* X_VECTOR[VECTOR_LENGTH] // THESE SHOULD ALSO BE
  FLOAT* Y_VECTOR[VECTOR_LENGTH] // GLOBAL
  VECTOR_LENGTH // GET FROM ARGS
  THREAD_COUNT = TC // GET FROM ARGS

  THREAD_HANDLES[]

  FOR(i=0; i<THREAD_COUNT; ++i)
   PTHREAD_CREATE(THREAD_HANDLES[i]; NULL; LOCAL_DOT(); i)

  FOR(i=0; i< THREAD_COUNT; ++i)
     PTHREAD_DESTROY(i)

  DESTROY_SEM(SEM)

  PRINT(DOT PRODUCT)
}
```

30/30

```
void* local_dot(void* rank) {
    long my_rank = (long) rank;
    int my_first_i = vector_length / thread_count * my_rank;
    int my_last_i = my_first_i + (vector_length / thread_count) - 1;
    float my_local_dot = 0;

    for (int i= my_first_i; i <= my_last_i; i++){
        my_local_dot += x_vector[i] * y_vector[i];
    }

    sem_wait(sem);
    global_dot += my_local_dot;
    sem_post(sem);

    return NULL;
}
```