Please print in pen:
Waterloo Student ID Number:

WatIAM/Quest Login Userid:

UNIVERSITY OF WATERLOO

Examination
Midterm
Winter 2016
ECE 459

Times: Friday 2016-03-04 at 18:30 to 19:30 (6:30 to 7:30PM)
Duration: 1 hour (60 minutes)
Exam ID: 3151084
Sections: ECE 459 LEC 001
Instructors: Jeffrey Zarnett

Open Book

Candidates may bring any reasonable aids. Open book, open notes. No electronics or communication devices.

Instructions:

1. This exam is open book, open notes (no electronics).
2. Turn off all communication devices.
3. There are four (4) questions, some with multiple parts. Not all are equally difficult.
4. The exam lasts 60 minutes and there are 50 marks.
5. If you feel like you need to ask a question, know that the most likely answer is "Read the Question". No questions are permitted. If you find that a question requires clarification, proceed by clearly stating any reasonable assumptions necessary to complete the question. If your assumptions are reasonable, they will be taken into account during grading.
6. Do not fail this city.
7. After reading and understanding the instructions, sign your name in the space provided below.

Signature

Marking Scheme (For Examiner Use Only):

Question	Mark	Weight
1		15
2a		10
2b		5
3		10
4		10
Total		**50**

Please initial:

Question 1: Manual Parallelization with pthreads [15 marks]

In lecture we saw some code similar to this that we wanted to parallelize:

```
#include <stdlib.h>
#define LENGTH (1024*1024)

int *vector;

void setup(int *vector, int length) {
    for (int i = 0; i < length; i++) {
        vector[i] += 1;
    }
}
int main() {
    vector = (int*) calloc(LENGTH, sizeof( int ));
    for (int i = 0; i < 1000; i++) {
        setup (vector, LENGTH);
    }
}
```

We discussed several strategies for manual parallelization. Using the pthreads library, parallelize this using the "good horizontal" strategy: create 4 threads; each thread does 1000 iterations on its own sub-array. You may modify the code in any way you deem necessary, as long as the code still does the same thing. Assume the C99 standard or higher is used (so variable declarations inside a for loop are allowed). Your solution should be free of race conditions and memory leaks.

For convenience, some of the pthread functions available for your use (and remember you can give `NULL` for the attributes and return values):

```
pthread_create( pthread_t *thread, const pthread_attr_t *attributes,
                void *(*start_routine)( void * ), void *argument )
pthread_join( pthread_t thread, void **returnValue )
pthread_cancel( pthread_t thread )
pthread_exit( void *value )
```

Complete the code below:

```
#include <stdlib.h>
#include <pthread.h>

#define LENGTH (1024*1024)
int *vector;
```

```
int main() {

    vector = (int*) calloc(LENGTH, sizeof( int ));
```

```
    pthread_exit( NULL );
}
```


Please initial:

Question 2: OpenMP [15 marks total]

2A: OpenMP Directives [10 marks]

Consider the following C program with OpenMP directives added:

```
int main (int argc, char *argv[])  {
int nthreads, i, tid;
float total;

#pragma omp parallel
  {
  tid = omp_get_thread_num();   /* Obtain thread number */

  if (tid == 0) { /* Only master thread does this */
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }
  printf("Thread %d is starting...\n",tid);

  #pragma omp barrier

  total = 0.0;
  #pragma omp for schedule(dynamic,10)
  for (i=0; i<1000000; i++)
     total = total + i*1.0;

  printf ("Thread %d is done! Total= %e\n",tid,total);
  }
}
```

When compiled and executed, it produces the following (incorrect) output:

```
Number of threads = 4
Thread 0 is starting...
Thread 1 is starting...
Thread 3 is starting...
Thread 2 is starting...
Thread 2 is done! Total= 0.000000e+00
Thread 2 is done! Total= 0.000000e+00
Thread 2 is done! Total= 0.000000e+00
Thread 2 is done! Total= 0.000000e+00
```

Explain what is wrong with this program, and how to fix it.

2B: OpenMP Reduction [5 marks]

In OpenMP we can do a reduction on a loop like the following:

```
#pragma omp parallel for reduction(+:sum)
  for (i=0; i < n; i++) {
    sum = sum + (a[i] * b[i]);
  }
```

This works when the reduction uses the mathematical addition ($+$) operator. Explain why we cannot specify an OpenMP reduction with the mathematical subtraction ($-$) operator.

Please initial:

Question 3: Short Answer [10 marks]

Answer these questions using at most three sentences. Each is worth 2 marks.

(a) Is there a situation where it is okay to allow two concurrent writes to the same variable without synchronization constructs like a mutex? Hint: Think about Assignment 1.

(b) Under what circumstances would it be appropriate to use spinlocks instead of a mutex or semaphore (or other synchronization construct)?

(c) Suppose you are running a Monte Carlo calculation on an otherwise unloaded 8-core machine. Why might performance decrease if you increase the number of threads from 8 to 9?

(d) What does the `volatile` qualifier do in C, and why is it unsuitable as a tool to avoid race conditions?

(e) When might the compiler decide that parallelization of a loop is "not profitable"?

Please initial:

Question 4: Cache Coherence [10 marks]

In lectures we discussed cache coherence with the MESI protocol (Modified/Exclusive/Shared/Invalid). Suppose we have 2 CPUs in the system, each with its own cache. Suppose there are data blocks `x` and `y` in the system and that cache has space for exactly two blocks. All cache lines are initially in the "I" (invalid) state.

The two CPUs issue the following instructions, in this order:

1. CPU0 reads `x`.
2. CPU1 reads `x`.
3. CPU0 writes `y`.
4. CPU1 writes `x`.
5. CPU1 reads `y`.

Complete the table below to reflect the MESI state of the cache of each CPU after each step in the above list has taken place. Use the letters M, E, S, and I to indicate the states of Modified, Exclusive, Shared, and Invalid, respectively. The actual values of `x` and `y` read and written are not relevant.

	CPU 0		**CPU 1**	
Step	**x**	**y**	**x**	**y**
0.	I	I	I	I
1.				
2.				
3.				
4.				
5.				

Please initial: