THE UNIVERSITY OF ALABAMA IN HUNTSVILLE
Department of Electrical and Computer Engineering

CPE 412/512 Exam 2

Fall Semester 2017
Exam 2 (Take-Home)

Name _____

INSTRUCTIONS: **CPE 512 students must work <u>all</u> five (5) problems. CPE 412 students should work any four (4) problems clearly indicating which problem they will omit. In addition to electronic submissions on the UAH Canvas course administration system student are required to submit a complete hardcopy of this exam by its due date at the beginning of class on Monday November 13, 2017.**

Statement of Compliance with UAH Academic Misconduct Policies

I _____ certify that I have worked independently of
(sign your name)
others on this test and the work that I am presenting is my own. I am familiar with the UAH academic misconduct policy as outlined in the UAH student handbook and have agreed to abide by the policies that are stated in this document.

Alternative Statement

I _____ am unwilling to sign the above statement of
(sign your name)
compliance because I cheated in some way on the problems listed below:

_____

1. *Fully answer the following questions. Justify your answers by providing the page/section number reference from your text which supports your answer or by providing a complete citation of any external sources that you have used.*

a) What OpenMP directive is used to provide mutual exclusion synchronization in sections of code? Give an example of its use and explain why it is needed.

b) Give the output of the following OpenMP code fragment if possible or discuss why this is not allowed. What is the effective scope of the sum variable? How could the answer be affected if the reduction clause is not used? Assume that there are 4 threads all together:

```
int i; double sum = 0.0;
#pragma omp parallel for reduction(+:sum) num_threads(4)
for (i=1; i <= 4; i++)
       sum = sum + i;
cout << "The sum is " << sum << endl;
```

*c) For the following **pthread** code what is (are) the possible value(s) of num if there are no assumptions regarding the targeted system or the manner in which the **pthread** scheduler will operate? Justify your answer.*

```
using namespace std;
#include <iostream>
#include <pthread.h>
#include <stdlib.h>

int num=1;
void *thread1(void *dummy) {
   num = num + 1;
}
void *thread2(void *dummy) {
   num = num * 3;
}
void *thread3(void *dummy) {
   num = 123;
}

int main(int argc, char *argv[]) {
   pthread_t thread1_id,thread2_id,thread3_id;
   pthread_create(&thread1_id,NULL,thread1,NULL);
   pthread_create(&thread2_id,NULL,thread2,NULL);
   pthread_create(&thread3_id,NULL,thread3,NULL);
   cout << "num = " << num << endl;
   while(1); // infinite loop
}
```

*d) What is the difference between the **ordered** directive and the **critical** section directives in OpenMP? Give separate examples of the use of both constructs. Which constructs might be useful to support local synchronization? Which constructs are useful for or global synchronization? Explain.*

*e) What is the difference between the **single** and **master** section directives in OpenMP? Give separate examples of the use of both constructs.*

f) Explain why static mapping of data blocks to processors may be bad for the Mandelbrot program that was discussed in one of the class lectures.

g) What is the basic idea of a Monte Carlo simulation. Describe how this technique can be used to solve the numerical integration problem? What makes this technique so attractive for parallel processing? What are the major issues that could affect its accuracy when Monte Carlo simulations are parallelized?

2. *You have been provided with a sequential exhaustive search traveling salesman program that you are to parallelize using OpenMP, or pThreads. This sequential program which is named* ***tsp_serial.cpp****, can be found on the CPE 512/412 Canvas™ site (or can be copied from /cpe412/exam2/tsp_serial.cpp on the UAH Jetson system).*

   a) Examine this program and modify it so that you can measure the execution time. Using the queuing system on the Jetson system, as discussed in class, record the execution times for a 3,4,5,6,7,8,9,10,11,12, and 13 city tour. What is the time complexity of this algorithm? Develop an equation that can be used to estimate the execution time of the base sequential algorithm. Why do you think this problem is a challenging problem to solve, and inexact heuristics are utilized instead of exhaustively searching through all solutions as is done in this program?

   b) Develop a general multi-threaded version of the sequential program in either pThreads or OpenMP that will effectively divide up the amount of work that is performed. Using a single node of the Jetson queuing system, measure the execution times for an 8,9,10,11,12, and 13 city tour on a 1, 2, 3, and 4 thread implementation. For each multi-threaded implementation show the speedup, efficiency, and cost as a function of the number of cities in the tour.

3. *Expand the* ***mm_mult_serial.cpp*** *program on the CPE 512/412 Canvas™ site (or copy this program from /cpe412/exam2/mm_mult_serial.cpp on the UAH Jetson system) to create a hybrid multi-threaded/message passing implementation of a matrix/matrix multiplication program where the first matrix is of size* ***lxm*** *and the second matrix is of size* ***mxn****. Assume that the quantities being multiplied are of type* ***float****. Write the program in a general manner to allows the number of message passing processes and the number of threads per message passing process to be independently set by the user at run time. The program should be designed using the same row-wise decomposition method that was used in the third homework assignment. It should be written in a manner that will result in the total amount of computation to be divided as evenly as possible among the message-passing processes with the computation that is assigned to each message-passing processes in turn being further divided as evenly as possible between the associated threads. You are to use a combination of* ***MPI*** *and either* ***OpenMP*** *or* ***pThreads*** *to complete this problem. The number of threads should be a command-line parameter while the number of message-passing processes should be set by the Jetson Queuing system when you specify the number of nodes that are to be employed. The l, m, and n dimensions should also be command line parameters that can be set at run time. In other words, if the executable is named* ***mm_mult_hybrid*** *then the syntax needed to execute the code should take on the general form shown below:*

   srun mm_mult_hybrid [No. Threads per Process] [dim_l] [dim_m] [dim_n])

   where the number of MPI processes to be employed which is specified when the job is submitted to the Jetson queuing system.

   a) Illustrate the correctness of your program for all possible combinations of thread numbers and number of MPI processes whose product is equal to 8 (i.e. 1 MPI process 8 threads per process, 2 MPI processes and 4 threads per process, 4 MPI process and 2 threads per process, and 1 MPI process and 8 threads per process) for cases where ***dim_l****,* ***dim_m****, and* ***dim_n*** *take on distinct values.*

   b) Then perform a set of timing experiments under the same set of MPI process/thread combinations of part a (but with the output suppressed) for the cases where the ***dim_l=dim_m=dim_n=****5,000****. Record the execution time associated with each multi-threaded case and compare these times with that of the original serial program. What is the relative speedup and efficiency for each case. Is there a significant difference between the various parallel implementations? If so give a possible explanation as to why the execution time was not the same for each case given that the total number of threads is the same in all cases and your code was designed to evenly distribute the workload among the threads.*

4. Answer the following question for the code segment shown below assuming that we have a computing node that has 4 active cores that are dedicated to processing our problem and that were are utilizing OpenMP to parallelize a **for**-loop that initializes the upper triangle portion of a 100 × 100 matrix to the values returned by the function, **init_element(x,y),** which itself always executes in constant amount of time regardless of the values associated with its two arguments.

```
#pragma openmp parallel for schedule( ... )
for (i = 0; i < 99; i++) {
      for (int j = i+1; j < 100; j++) {
            a[i][j] = init_element(i,j);
      }
}
```

Notice that the arguments to the schedule() clause have been left undefined. Below are six example schedule clauses that could be used. **Rank these clauses from slowest to fasted** by closely examining the characteristics of this problem. To do this note that each iteration of the inner loop above does just one assignment and we can estimate the execution time by counting how many assignments each thread does. (Also note the total number of assignments the problem performs is exactly 4,950 assignments). **For each schedule clause, estimate how long the parallelized loop will run. Explain how you arrived at your estimates.**

schedule(static)

schedule(static, 10)

schedule(static, 1)

schedule(dynamic, 1)

schedule(dynamic, 10)

schedule(dynamic, 20)

The schedule clause affects how loop iterations are mapped onto threads

`schedule(static [,chunk])`

- Blocks of iterations of size "chunk" to threads
- Round robin distribution
- Low overhead, may cause load imbalance

`schedule(dynamic[,chunk])`

- Threads grab "chunk" iterations
- When done with iterations, thread requests next set
- Can reduce load imbalance, Significant Overhead involved compared to static scheduling.

5. *Expand upon the **bounded_buffer.cpp** that is provided on Canvas (or copy this program from /cpe412/exam2/bounded_buffer.cpp on the Jetson system) to create two separate OpenMP implementations of the producer/consumer bounded buffer-problem discussed in class that utilizes a common shared memory (between threads) and counting semaphores to ensure proper synchronization. [Note you will also need to link this program with the **util.o** file that is also provided on Canvas or on the Jetson system at /cpe412/exam2/util.o] One implementation should use OpenMP's standard parallel and work sharing data parallel constructs to start up the separate consumer and producer threads. The other implementation should utilize the OpenMP tasking model. Verify that both models function correctly and answer the following questions:*

   a) Carefully explain the characteristics of the producer and consumer bounded buffer problem? What are the major synchronization challenges that are involved.

   b) Describe the function of the main, consumer, and producer module? What are the output files diary and con_? and prod_? reporting?

   c) Execute both versions of the producer/consumer program with two producers, two consumers, number of messages of 100 and with a buffer size of 5. How many actual threads are generated when you run each version? Are they the same? How do they relate to the number of producers/consumers that are specified at run time?

   d) How is it possible for the buffer size to be smaller than the number of data that is to be passed between producer and consumer? Explain how this is actually implemented in the program.

   e) Experiment with different size buffers, messages, and consumer processes and producer processes. From the diary file and other files can you determine if the semaphore releasing strategy is fair? Explain your answer.