Instructions.

1. Attempt all questions. Partial credit will be given.
2. Show all the steps of your work clearly.
3. No digital media allowed (phones, laptops, etc.)

<table>
<thead>
<tr>
<th>QUESTION</th>
<th>POINTS</th>
<th>YOUR SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>
[Q1] (12 points):

(a) (2 points): Define and describe the term Pthreads:
Posix Threads; Posix (Parallel Operating System Interface); shared memory; light weight processes; used for parallel jobs in place of fork(); parallel threads; developed for HPC applications; more features that Pthreads.

(b) (2 points): Define and describe the term OpenMP:
Open Multiprocessing; shared memory; multi threads; multi-platform shared-memory parallel programming; split tasks to run in parallel.

(c) (2 points): What do Pthreads and OpenMP have in common?
shared memory; multi-threads; lightweight processes; easy to implement;

(d) (2 points): What are the limitations of the thread programming model?
sychronization; limited to # of cores on a node; scaleability 1-2 threads/core.

(e) (2 points): Which model (Pthread or OpenMP) is better for scientific computing?
OpenMP – should have some explanation: contains API with more support for computing, barriers, etc.

(f) (2 points): For general multithread programming, what does scope mean for shared and global variables?
In shared memory, most variables are shared by default. Variable scope is defined by model: Pthread or OpenMP. In general: Global - variables that are accessible by all threads and main during application lifetime; shared variables are created or visible during thread lifetime so scope can be limited by different thread mechanisms.
[Q2] (10 points): Two programs are listed below in Code Block A and Code Block B. They are different programs. Refer to these blocks of code when answering the questions on this page.

Code Block A

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

int thread_count;
void *Hello(void* rank); /* Thread function */

int main(int argc, char* argv[]) {
    long thread;
    pthread_t* thd_hndle;
    thread_count = strtol(argv[1], NULL, 10);
    thd_hndle = malloc (thread_count*sizeof(pthread_t));

    for (thread = 0; thread < thread_count; thread++)
        pthread_create(&thd_hndle[thread], NULL, Hello, (void*) thread);

    printf("Hello from the main thread\n");

    for (thread = 0; thread < thread_count; thread++)
        pthread_join(thd_hndle[thread], NULL);

    free(thd_hndle);
    return 0;
}

void *Hello(void* rank) {
    long my_rank = (long) rank;
    printf("Hello from thread %ld of %d\n", my_rank, thread_count);
    return NULL;
}
```

Code Block B

```c
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char *argv[]) {
    int nthds, thd_rank;
    int thdsum=0;

    #pragma omp parallel default(shared) \ 
       private(thd_rank, nthds)
    {
        nthds = omp_get_num_threads();
        thd_rank = omp_get_thread_num();
        printf("Hello from thread %d out of %d\n", thd_rank, nthds);
        thdsum += thd_rank;
    }

    printf("Nthreads=%d, ThreadSum=%d\n",nthds,thdsum);
    return 0;
}
```

(a) (1 point): Which type of thread programming model is Block A? Pthreads

(b) (1 point): Which type of thread programming model is Block B? OpenMP

(c) (2 points): Which line(s) launch the threads in in Block A and how are they set?
   lines 14-15
   line 10: command line argument

(d) (2 points): Which line(s) launch the threads in in Block B and how are they set?
   lines 9-10 pragma call
   OMP_NUM_THREADS

(e) (2 points): Write the output for line 28 in Block A for 3 threads.
   Hello from thread 2 of 3
   Hello from thread 0 of 3
   Hello from thread 1 of 3

(f) (2 points): Write the output for line 20 in Block B for 3 threads.
   lines 14-15:
   Hello from thread 1 out of 3
   Hello from thread 2 out of 3
   Hello from thread 0 out of 3
   line 20:
   Nthreads=3, ThreadSum=3
[Q3] (14 points): The code shown below in Blocks A and B both estimate the value of \( \pi \). Use these blocks of code to answer the questions on this page.

**Code Block A**
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

int main(int argc, char* argv[]) {
    long long n, i;
    int thread_count;
    double factor;
    double sum = 0.0;

    thread_count = strtol(argv[1], NULL, 10);
    n = strtoll(argv[2], NULL, 10);

    #pragma omp parallel for num_threads(thread_count) reduction(+: sum) private(factor)
    for (i = 0; i < n; i++) {
        factor = (i % 2 == 0) ? 1.0 : -1.0;
        sum += factor/(2*i+1);
        printf("Thread %d > i = %lld, my_sum = %f\n",
               my_rank, i, my_sum);
    }
    sum = 4.0*sum;
    printf(" N=%lld, #Thds=%d, PIest = %.14f, PIref= %.14f\n",
            n, thread_count,sum, 4.0*atan(1.0));
    return 0;
}
```

**Code Block B**
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

int main(int argc, char* argv[]) {
    long long n, i;
    int thd_cnt;
    double factor;
    double sum = 0.0;
    thd_cnt = strtol(argv[1], NULL, 10);
    n = strtoll(argv[2], NULL, 10);

    #pragma omp parallel num_threads(thd_cnt)
    {
        int my_rank = omp_get_thread_num();
        int start=tid*(n/thd_cnt), end=start + n/thd_cnt -1;
        double my_sum=0;
        int i;
        for (i = start; i <= end; i++) {
            factor = (i % 2 == 0) ? 1.0 : -1.0;
            my_sum += factor/(2*i+1);
            printf("Thread %d > i = %lld, sum = %f\n",
                   my_rank, i, my_sum);
        }
    }
    sum += my_sum;
    sum = 4.0*sum;
    printf(" N=%lld, #Thds=%d, PIest = %.14f, PIref= %.14f\n",
            n, thread_count,sum, 4.0*atan(1.0));
    return 0;
}
```

(a) (1 points): What method of thread programming model is used in the code above? 
**OpenMP, shared memory**

(b) (3 points): Name 3 methods for controlling access to critical blocks for this method? 
**parallel, parallel for, atomic, critical, omp locks,...**

(c) (4 points): What line(s) define the critical section and what method is used? (list or highlight) 
**Block A: lines 15-16: omp parallel for; reduction operator**
**Block B: lines 15: omp parallel;**

(d) (6 points): Explain the differences between the two methods used in Blocks A and B: 
**Block A uses parallel for loop and reduction operator which does two things:**
* parallelizes the thread work and divides it into chunks to be done by each thread
* protects the iterate variable, i, and the summation variable, sum.
**Block B uses parallel block and a for loop. It does not work correctly:**
* the iterate variable, i, and the summation variable, sum are not protected and are shared by all threads.
* the result is wrong and unrealistic.

```
omp_pi_err 8 1500
For N=1500, #Thds=8, Piest = 3.14092598699720, PIref= 3.14159265358979
```
[Q4] (14 points): The code shown below in Blocks A and B uses a Maclaurin series to estimate π. Use these blocks of code to answer the questions on this page.

**Block A**

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
long thd_cnt, n;
int flag;
double sum;
void* Thread_sum(void* rank);
int main(int argc, char* argv[])
{
    long thread;
    pthread_t* thd_hndles;
    thd_cnt = strtol(argv[1], NULL, 10);
    n = strtoll(argv[2], NULL, 10);
    thd_hndles = (pthread_t*) malloc (thd_cnt*sizeof(pthread_t));
    sum = 0.0; flag = 0;
    for (thread = 0; thread < thd_cnt; thread++)
        pthread_create(&thd_hndles[thread], NULL,
            Thread_sum, (void*)thread);
    for (thread = 0; thread < thd_cnt; thread++)
        pthread_join(thd_hndles[thread], NULL);
    sum = 4.0*sum;
    printf(" For #Thds= %d, N= %lld terms, Multi-threaded estimate of pi = %.15f
", thd_cnt, n, sum);
    free(thd_hndles);
    return 0;
} /* main */
```

**Block B**

```c
void* Thread_sum(void* rank) {
    long i, my_rank = (long) rank;
    double factor, my_sum = 0.0;
    long long my_n = n/thd_cnt;
    for (i = my_first_i; i < my_last_i; i++, factor = -factor)
        my_sum += factor/(2*i+1);
    while (flag != my_rank);
    sum += my_sum;
    flag = (flag+1) % thd_cnt;
    return NULL;
} /* Thread_sum */
```

(a) (2 points): What method of thread programming model is used in the code above?

*Pthreads, shared memory*

(b) (2 points): What is the scope of the variables `thd_cnt` and `n`, and why?

*these are defined outside of main, so they are global and used in Thread_sum*

(c) (3 points): Name 3 methods for controlling access to critical sections for this method?

*busy wait, mutex, semaphore, barrier, condition vars*

(d) (2 points): What line(s) define the critical section and what method is used? (list or highlight)

*Block B, lines 16-18; using busy_wait on the flag state.*

(e) (5 points): Rewrite the code/critical section to use one of the other methods you listed above.

**Semaphore:**

```c
sem_t sem;
main {
    ... sem_init(&sem, 0, 1);
    pthread_create(&thd_hndles[thread], NULL, Thread_sum, (void*)thread);
    ...}

void* Thread_sum(void* rank) {
    ... compute my_sum;
    sem_wait(&sem);
    sum += my_sum;
    sem_post(&sem);
    ...}

} /* Thread_sum */
```

**Mutex:**

```c
pthread_mutex_t mutex;
main {
    ... pthread_create(&thd_hndles[thread], NULL, Thread_sum, (void*)thread);
    ...}

void* Thread_sum(void* rank) {
    ... compute my_sum;
    pthread_mutex_lock(&mutex);
    sum += my_sum;
    pthread_mutex_unlock(&mutex);
    ...}

} /* Thread_sum */
```