CS 160: Lab/Assignment 4

Due at 11:59PM on March 14, 2017

1. Create a parallel version of the pi program using a ***parallel construct***. Please use the OpenMP runtime library routine omp_get_wtime() to measure the execution time of the computational section in the program.

2. Create a parallel version of the pi program using a ***loop construct.*** Your goal is to minimize the number changes made to the serial program. Please use the OpenMP runtime library routine omp_get_wtime() to measure the execution time of the computational section in the program.

3. Parallelize the matrix multiplication program in the file matmul.c attached. Can you optimize the program by playing with how the loops are scheduled? Please use the OpenMP runtime library routine omp_get_wtime() to measure the execution time of the computational section in the program.

4. Does the following OpenMP code segment parallelize the for-loop correctly or not? Why?

```
int i, j,  a[MAX];
j=1;
#pragma omp parallel for
for (i=0; i<MAX; i++) {
     j=j+2;
     a[i]=comp(j);
}
```

5. Consider the following OpenMP program segment.

```
int a=1, b=2, c=3, d=4;
…
#pagama omp parallel private(b), firstprivate(c) lastprivate(d)
{
…
}
```

   (a). Are a, b, c, and d local or shared in the parallel region?
   (b). What are their initial values inside the parallel region?

6. The goal of the following OpenMP program is to calculate π in parallel. Which variables are shared and which variables are private in the parallel region of the program? Identify and fix all bugs in the program.

```
#include <stdio.h>
#define MAX_THREADS 4
static long num_steps = 100000000;
double step;
int main ()
{
            int i, j;
            double pi, full_sum = 0.0;
            double start_time, run_time, x;
            step = 1.0/(double) num_steps;
       for(j=1; j<=MAX_THREADS; j++){
             omp_set_num_threads(j);
             full_sum = 0.0;
                  start_time = omp_get_wtime();
#pragma omp parallel
{
                  int id = omp_get_thread_num();
                  int numthreads = omp_get_num_threads();
                  double partial_sum = 0;
                  for (i=id; i< num_steps; i+=numthreads){
                              x = (i+0.5)*step;
                              partial_sum = partial_sum + 4.0/(1.0+x*x);
                  }
                      full_sum += partial_sum;
}
          pi = step * full_sum;
               run_time = omp_get_wtime() - start_time;
               printf("\n pi is %f in %f seconds %d threads \n ", pi,
run_time, j);
     }
}
```