CS 160: Lab/Assignment 4

Due at 11:59PM on March 14, 2017

Numbers 1 and 2 use slide 26 in the openmp slides
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.

Please see num1.c
num_steps = 100000
Threads: 4
Execution time: 0.004686 seconds

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.

Please see num2.c
num_steps = 100000
Threads: 4
Execution time: 0.010468 seconds

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.

Yes, the program can be optimized for stronger hardware. If there is a CPU bottleneck, however, parallelization takes more time.

Execution time:
The sequential version of matrix multiplication costs 4.904458 seconds
The parallel version of matrix multiplication costs 1.901743 seconds

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);
}
```

No, the following OpenMP code does not segment parallelize the for-loop correctly. Each thread should have different values for j, but in this case, they start out with the same value for j because it is defined as the integer 1 outside the OpenMP code segment. The whole point of parallelized loops is so they can safely execute in any order without loop-carried dependencies, but the above code does not do this.

5. Consider the following OpenMP program segment.

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

(a). Are a, b, c, and d local or shared in the parallel region?
The variable a is shared.
The variable b is local.
The variable c is local.
The variable d is local.

(b). What are their initial values inside the parallel region?
Initial Value of a is 1
Initial Value of b is garbage (not initialized)
Initial Value of c is 3
Initial Value of d is garbage (not initialized)

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 private(x), private(i)
          {
               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);
     }
}
```

In the code provided by the assignment...

Private variables: id, numthreads, partial_sum, i
Shared variables: x, full_sum