ECE 420 Parallel and Distributed Programming Assignment 2

Instructor: Di Niu
Email: dniu@ualberta.ca
Department of Electrical and Computer Engineering
University of Alberta

Due date: see the schedule posted on course website

Note: this assignment provides some sample questions that are representative of the questions to appear in exams, although questions in exams will be asked in a more formal and rigorous way. The marking of this assignment is largely based on efforts. Solutions will be posted after the due date.

1. In the serial Trapezoidal Rule calculation, we have a for loop containing 1-9999 iterations, each of which calculates the area of a stripe. Then the areas are summed up to give the total approximate area under the function of interest. Suppose now we want to parallelize this for loop with OpenMP using 2 threads. What are the iterations assigned to Thread 0 and Thread 1, respectively, when the clause `schedule(static,2)` is used? What are the iterations assigned to Thread 0 and Thread 1, respectively, when the clause `schedule(guided)` is used?

Note: we assume each iteration takes a constant amount of time and ignore all the scheduling/ synchronization/work assignment overhead associated with the dynamic assignment of work.

2. An OpenMP solution to the Odd-Even Transposition Sort is given in the following program. Explain why this program might be inefficient. Modify the program to reduce the run time.

```
   for (phase = 0; phase < n; phase++) {
      if (phase % 2 == 0)
#        pragma omp parallel for num_threads(thread_count) \
            default(none) shared(a, n) private(i, tmp)
         for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
               tmp = a[i-1];
               a[i-1] = a[i];
               a[i] = tmp;
            }
         }
      else
#        pragma omp parallel for num_threads(thread_count) \
            default(none) shared(a, n) private(i, tmp)
         for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
               tmp = a[i+1];
               a[i+1] = a[i];
               a[i] = tmp;
            }
         }
   }
```

3. The following program finds the maximum value from an array `data`. How can we modify it to improve the performance?

```
int largest = 0;
#pragma omp parallel for
for ( int i = 0; i < 1000; i++ ) {
  #pragma omp critical
  if (data[i] > largest)
     largest = data[i];
}
```

4. The following two program segments implement matrix-vector multiplication, parallelizing it by rows and by columns, respectively.

```
// Parallelize by rows
# pragma omp parallel default(none) shared(v2,v1,matrix,tam) private(i,j)
{
#   pragma omp for
    for (i = 0; i < tam; i++)
        for (j = 0; j < tam; j++)
            v2[i] += matrix[i][j] * v1[j];
}

// Parallelize by columns
# pragma omp parallel default(none) shared(j,v2,v1,matrix,tam) private(i,j)
{
      for (i = 0; i < tam; i++)
#           pragma omp for
            for (j = 0; j < tam; j++)
                  v2[i] += matrix[i][j] * v1[j];
}
```

Are these programs correct? If not, please fix the problems.

5. What's a possible output result of the following program?

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

int main()
{
    omp_set_nested(1);
    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2)
    {
        if (omp_get_thread_num() == 0)
            omp_set_num_threads(4);
        else
            omp_set_num_threads(6);

        printf("%d: %d %d\n", omp_get_thread_num(),
               omp_get_num_threads(),
               omp_get_max_threads());

        #pragma omp parallel
        {
```

```
            #pragma omp master
            {
                printf("Inner: %d\n", omp_get_num_threads());
            }
            omp_set_num_threads(7);
        }

        #pragma omp parallel
        {
            printf("count me.\n");
        }
    }
    return(0);
}
```

6. The following is a piece of recursive C code to calculate a Fibonacci number for $n = 5$:

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int fib(int n) {
      if (n<2)
            return n;
      else {
            return fib(n-1) + fib(n-2);
      }
}

int main (int argc, char *argv[])
{
      int result;
      printf("Doing sequential fibonacci:\n");
      result = fib(5);
      printf("Result is %d\n", result);
      return 0;
}
```

1) Parallelize the program above using OpenMP Sections directives.
2) Parallelize the program above using OpenMP Tasks directives.
3) If the following environment variables are set, what are the number of threads that are ever launched in the parallel programs 1) and 2), respectively? Assume no `num_threads` clause or OpenMP functions were used in your program.

```
export OMP_NUM_THREADS = 3
export OMP_NESTED = TRUE
export OMP_DYNAMIC = FALSE
```