E517/317 – Introduction to High Performance Computing
Fall 2019
Assignment 2

1. Can you spot any mistakes in the following code? Please correct them and submit the corrections in code form.

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

// compute the dot product of two vectors

int main() {
  int const N=100;
  int i, k;
  double a[N], b[N];
  double dot_prod = 0.0;

  int thread_id;
  // Arbitrarily initialize vectors a and b
  for(i = 0; i < N; i++) {
    a[i] = 3.14;
    b[i] = 6.67;
  }

  #pragma omp parallel private(thread_id)
  {
    thread_id = omp_get_thread_num();
    printf("This thread is: %d\n", thread_id);
    #pragma omp for
    for(i = 0; i<N; i++){
      // sum up the element-wise product of the two arrays
      dot_prod = dot_prod + a[i] * b[i];
    }
  }

  printf("Dot product of the two vectors is %g\n", dot_prod);

  return 0;
}
```

2. In line 22 of the following code, the static scheduler is demonstrated.

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 int main (int argc, char *argv[])
 6 {
 7   const int N = 38;
 8   int nthreads, threadid, i;
 9   double a[N], b[N], result[N];
10
11   // Initialize
12   for (i=0; i < N; i++) {
13     a[i] = 1.0*i;
14     b[i] = 2.0*i;
15   }
16
17   int chunk = 7;
18   #pragma omp parallel private(threadid)
19   { // fork
20     threadid = omp_get_thread_num();
21
22     #pragma omp for schedule(static, chunk)
23     for (i=0; i<N; i++) {
24       result[i] = a[i] + b[i];
25       printf(" Thread id: %d working on index %d\n",threadid,i);
26     }
27
28   } // join
29
30   printf(" TEST result[19] = %g\n",result[19]);
31
32   return 0;
33 }
```

How would the output of this code change if the dynamic scheduler were used instead?

3. In the following code, the sections pragma is presented.

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

int main()
{
  const int N = 1000;
  int x[N], i, max_x, min_x,sum,sum2;
  float mean, mean2, var;
  sum = 0;
  sum2 = 0;
  max_x = 0;
  min_x = 100;

#pragma omp parallel for
  for(i = 0; i < N; i++) {
    x[i] = i;
  }

#pragma omp parallel private(i) shared(x)
{
// Fork several different threads
#pragma omp sections
  {
      {
        for(i = 0; i < N; i++) {
          if (x[i] > max_x) max_x = x[i];
          if (x[i] < min_x) min_x = x[i];
        }
        printf("The max of x = %d\n", max_x);
        printf("The min of x = %d\n", min_x);
      }
#pragma omp section
      { // Calculate the mean of x
        for(i = 0; i < N; i++)
          sum = sum + x[i];
        mean = sum/N;
        printf("Mean of x = %f\n", mean);
      }
#pragma omp section
      {
        // Calculate the sum of the squares of x
        for(i = 0; i < N; i++)
          sum2 = sum2 + x[i]*x[i];
        mean2 = sum2/N;

      }
  }
}

  var = mean2 - mean*mean;
  printf("Variance of x = %f\n", var);
  return 0;
}
```

(a) What prints to screen if this code is run on 3 OpenMP threads?
(b) What prints to screen if this code is run on 5 OpenMP threads?
(c) What prints to screen if this code is run on 1 OpenMP thread?
(d) In general, how does the number of sections impact the choice of number of OpenMP threads?

4. Modify the following serial matrix-vector code and add in OpenMP. Plot the strong scaling.

```c
#include <stdio.h>
// link with -lm at compile time
#include <math.h>


int main()
{
  const int N = 1000;
  int i, j;

  double A[N*N];
  double x[N], b[N];

  // initialize the matrix and the vector
  for(i=0;i<N;i++){
    for(j=0;j<N;j++){
      A[i*N + j] = sin(0.01*(i*N + j));
    }
    b[i] = cos(0.01*i);
    x[i] = 0.0;
  }

  //matrix vector multiplication
  for(i=0;i<N;i++){
    for(j=0;j<N;j++){
      x[i] += A[i*N + j]*b[j];
    }
  }

  printf("x[%d] = %g\n",505,x[505]);

  return 0;
}
```