E517 – Introduction to High Performance Computing
Assignment 2

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

```
#include <stdio.h>
#include <omp.h> /* provide OpenMP runtime libraries */

/* Calculate inner product of two vectors */

int main() {
  const int N = 100; /* initialize vector size */
  int i, k; /* declare index i and k and bound N */
  float a[N], b[N]; /* declare vectors a and b */
  float dot_prod; /* declare result value dot_prod */
  dot_prod = 0.0; /* initialize result value */

  for(k = 0; k < N; k++) { /* initialize a and b */
    a[k] = 3.0 * k;
    b[k] = 1.8 * k;
  }

#pragma omp parallel
{
#pragma omp for
  for(i = 0; i < N; i++) { /* compute dot product */
    dot_prod = dot_prod + a[i] * b[i];
  }
}

  printf("Inner product of a[] and b[] = %f\n", dot_prod);

  return 0;
}
```

2. In line 23 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 = 28;
 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
19 #pragma omp parallel private(threadid)
20   { // fork
21   threadid = omp_get_thread_num();
22
23   #pragma omp for schedule(static,chunk)
24   for (i=0; i<N; i++) {
25     result[i] = a[i] + b[i];
26     printf(" Thread id: %d working on index %d\n",threadid,i);
27   }
28
29   } // join
30
31   printf(" TEST result[19] = %g\n",result[19]);
32
33   return 0;
34 }
```

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.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4 #include <omp.h>
 5
 6 int main()
 7 {
 8   const int N = 1000;
 9   int x[N], i, max_x, min_x, sum,sum2;
10   float mean,mean2,var;
11   max_x = 0;
12   min_x = 100;
13   sum = 0;
14   sum2 = 0;
15
16   /* initialize x */
17   srand(1.0); // Initialize random variable seed
18 #pragma omp parallel for
19   for(i = 0; i < N; i++) {
20     x[i] = rand();
21   }
22
23
24 #pragma omp parallel private(i) shared(x)
25 {
26 #pragma omp sections
27   {
28   /* fork 3 different threads */
29       {
30         for(i = 0; i < N; i++) { /* find min & max of x */
31           if (x[i] > max_x) max_x = x[i];
32           if (x[i] < min_x) min_x = x[i];
33         }
34         printf("The max of x = %d\n", max_x);
35         printf("The min of x = %d\n", min_x);
36       }
37 #pragma omp section
38       { /* calculate the mean of x */
39         for(i = 0; i < N; i++)
40           sum = sum + x[i];
41         mean = sum/N;
42         printf("Mean of x = %f\n", mean);
43       }
44 #pragma omp section
45       {
46         for(i = 0; i < N; i++)
47           sum2 = sum2 + x[i]*x[i];
48         mean2 = sum2/N;
49       }
50   }
51 }
52   var = mean2 - mean*mean;
53   printf("variance of x = %f\n",var);
54   return 0;
55 }
```

(a) What prints to screen if this code is run on 5 OpenMP threads?
(b) What prints to screen if this code is run on 1 OpenMP thread?
(c) 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.

```
#include <stdio.h>
// Remember to link this code with -lm (it needs the math library)
#include <math.h>

int main() {

  const int size = 10000;
  int i,j;

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

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

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

  printf(" x[%d] = %g\n",5050,x[5050]);

  return 0;
}
```