Assignment 4 Question 1

Jonathan Gillett

Part A

i

The following pragma is applicable for parallelization because each value of k is independent and block can execute for different values of k in any order.

```
# pragma omp parallel for
for (k=0; k < (int) sqrt(x); k++) {
  a[k] = 2.3 * k;
  if (k<10) b[k] = a[k];
}
```

ii

The following may have issues if the blocks cannot execute at different orders (e.g. data is already partially sorted) and needs to be calculated up until $a[k] < b[k]$ and stop when no longer sorted. Otherwise there is no issue as flag is only being set to 1, there is no risk or a race condition.

```
flag = 0;
# pragma omp parallel for
for (k=0; (k<n) & (!flag); k++) {
  a[k] = 2.3 * k;
  if (a[k]<b[k]) flag = 1;
}
```

iii

The value of k is independent and block can execute for different values of k in any order.

```
# pragma omp parallel for
for (k=0; k<n; k++)
  a[k] = foo(k);
```

iv

The value of k is independent and the block can execute for different values of k in any order.

```
#pragma omp parallel for
for (k=0; k<n; k++) {
  a[k] = foo(k);
  if (a[k]<b[k]) a[k] = b[k];
}
```

v

The value of k is independent and block can execute for different values of k in any order.

```
#pragma omp parallel for
for (k=0; k<n; k++) {
  a[k] = foo(k);
  if (a[k]<b[k]) break;
}
```

vi

The OpenMP reduction operation is used for the *dotp* variable in order to properly perform the reduction summation in parallel.

```
dotp = 0.0;
#pragma omp parallel for reduction(+: dotp)
for (k=0; k<n; k++)
  dotp += a[k] * b[k];
```

vii

The following is not possible to parallelize due to the recurrence. It is not suitable for parallelization as the calculation of each $a[i]$ element is dependent on the results of the previous calculation for $a[i-k]$.

```
for (i=k; i<2*k; i++)
  a[i] = a[i] + a[i-k];
```

viii

The following is not possible to parallelize due to the recurrence. It is not suitable for parallelization as the calculation of each $a[i]$ element is dependent on the results of the previous calculation for $a[i-k]$.

```
for (i=k; i<n; i++)
  a[i] = b * a[i-k];
```

Part B

The code below could have parallelism for the nested loop as well, but it is *highly* impractical as the computation for the first for loop only involves initializing the $rowterm[i]$ and $colterm[i]$ entries. The following code below expresses as much parallelism for the loops below with as little unnecessary overhead as possible, for each nested loop the j index is made private to prevent any race condition errors from the index being overwritten by separate threads.

```
#pragma omp parallel for private(j)
for (i = 0; i < m; i++) {
  rowterm[i] = 0.0;
  for (j = 0; j < p; j++) {
    rowterm[i] += a[i][2*j] * a[i][2*j+1];
  }
}

#pragma omp parallel for private(j)
for (i = 0; i < q; i++) {
  colterm[i] = 0.0;
  for (j = 0; j < p; j++) {
    colterm[i] += b[2*j][i] * b[2*j+1][i];
  }
}
```