Problem 1

The serial program:

```
for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }
```

The parallel progarm:

```
#pragma omp parallel for num_threads(thread_count) private(i, j, k, sum)
shared(A, B, C_parallel)
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += A[i][k] * B[k][j];
            }
            C_parallel[i][j] = sum;
        }
    }
```

Result:

```
0.002000,0.001000
```

Problem 2

The serial program:

```
void serial_histogram(float *array, int n, int *bins, int num_bins)
{
    int i;
    /* Initialize the bins as zero */
    for (i = 0; i < num_bins; i++) {
        bins[i] = 0;
    }
    /* Counting */
    int idx;
    for (i = 0; i < n; i++) {
        int val = (int)array[i];
        if (val == num_bins) { /* Ensure 10 numbers go to the last bin */
            idx = num_bins - 1;
        } else {
            idx = val % num_bins;
```

```
        }
        bins[idx]++;
    }
    for(i = 0; i < num_bins; i++){
        printf("%d,", bins[i]);
    }
}
```

The parallel program:

```
void parallel_histogram(float *array, int n, int *bins, int num_bins, int thread_count) {
    int i;

    /* Initialize bins to zero */

    for (i = 0; i < num_bins; i++) {
        bins[i] = 0;
    }

    /* Thread-private bins to avoid race conditions */
    int idx;

    /* Parallel counting */
    #pragma omp parallel for num_threads(thread_count) reduction(+:bins[:num_bins]) private(idx)
    for (i = 0; i < n; i++) {
        int val = (int)array[i];
        idx = (val == num_bins) ? num_bins - 1 : val % num_bins;
        bins[idx]++;
    }

}
```

Result:

```
1000704,999293,1000213,1000989,999974,999431,1000450,999340,999259,1000347,0.110000,0.014000
```