Q1.1 The first error lies in the following line:

```
int threadID = *((int*)vargs);
```

The problem is that *vargs* points to the address of the counter in the for loop creating the threads. This means that by the time the thread tries to read from this memory location, it's value might have already changed. *threadID* should also be a long instead of an int.

Q1.2 The second error lies in the following line:

```
pi += 4.0 / double(N);
```

The problem is that the multiple threads all write to the same memory location, the variable pi.

Extra We also notice that the *cout* results in a data race condition. We fix this with a separate mutex so we don't disrupt the other calculations. Another way to fix this is with use of a buffer containing the entire string that has to be printed instead of using the <<-operator.

Q2.1

Before modifications

threads	runtime	speedup
1	0.33	1
2	0.425	0.79
4	0.4225	0.78
8	0.2265	1.46
16	0.23625	1.40

After modifications

threads	runtime	speedup
1	0.33	1
2	0.165	2
4	0.085	3.88
8	0.04375	7.54
16	0.023125	14.27

Q2.2

Before modifications

threads	runtime	speedup
1	0.13	1
2	0.06	2.17
4	0.0325	4.0
8	0.0215	6.05
16	0.01875	6.93

After modifications

threads	runtime	speedup
1	0.13	1
2	0.065	2.00
4	0.0325	4.00
8	0.02125	6.12
16	0.01825	7.12

Q2.3 The problem is that we're dealing with false sharing of cache. *localSum* is an array of doubles, this means that these doubles are stored next to each other in memory. This results in multiple of these doubles being in one and the same cache line. As we read from the different positions in this array we evict the cache lines of the other threads resulting in poor performance and especially, poor scaling.