Offload Practical

1. Hello Offload

First problem is simply a hello world program that prints "hello world", prints the number of processors there are, and then calculates PI using OpenMP threads.

1. STDOUT in offload is piped back to STDOUT on the host. Print "Hello world" from the MIC.
2. Offload the PI calculation.
3. Write a function that prints the hello world message for mic, if it is ran on the mic, and the hello world message for the host if it is ran on the host.

2. Function Offload

This exercise shows how to declare functions and variables as offloadable using Intel LEO.

1. Offload the `CountNonZero` calculation.
2. Part2 is a variant of the code in part1 that uses global variables and another compilation unit. Offload the CountNonZero calculation.

3. Data Persistence

This exercise covers offloading dynamically allocated arrays and also keeping them on the coprocessor after the offload has finished. Array p is dynamically allocated in the host code. The code calculates the sum of all the elements in p.

1. Offload the array sum calculation.
2. Allocate and offload the array p using offload_transfer. Free it using `offload_transfer` after the main offload.
3. Use a separate `offload_transfer` to retieve the value of sum after the main offload. Print the value of sum on the host after the offload and then after the offload_transfer.

4. Matrix

This practical multiplies a matrix with a vector.

1. Part1 uses stack arrays. Offload the matrix vector calculation. What happens if you don't explicitly list 'in/out/inout' clauses here?

2. Part2 is the same as part1 except it uses a dynamic arrays.
3. This example multiplies the matrix and vector `maxIter` times. Offload the matrix vector calculation inside the `iter` loop.
4. For each iteration execute part of the calculation on the MIC while executing the other part on the host concurrently. Combine the answers on the host.