1. In this problem, you will utilize the IEEE 754 format and evaluate the performance implications of using floats versus doubles in a computation.

a) Compute $f(x) = sin(x)$ using a Taylor series expansion. To refresh your memory:

$$sin(x) = \Sigma_0^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \cdots$$

You select the number of terms you want to compute (but at least 10 terms).
Compute $sin(x)$ for 4 different values, though be carful not to use too large a value.
Generate two versions of your code, first defining x and $sin(x)$ to use floats (SP), and second, defining them as doubles (DP). Discuss any differences you find in your results for $f(x)$. You should provide an in-depth discussion on the results you get and the reasons for any differences.

System: AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz

I choose 10 terms to compute and 1.0, 2.0, 3.0, 4.0 as values.

```
10
The single precision floating point function results of 1.000000 is: 0.8414709568023681640625000
The single precision floating point cmath results of 1.000000 is: 0.8414709568023681640625000
The single precision floating point function results of 2.000000 is: 0.9092974066734313964843750
The single precision floating point cmath results of 2.000000 is: 0.9092974066734313964843750
The single precision floating point function results of 3.000000 is: 0.1411200910806655883789062
The single precision floating point cmath results of 3.000000 is: 0.1411200016736984252929688
The single precision floating point function results of 4.000000 is: -0.7568024992942810058593750
The single precision floating point cmath results of 4.000000 is: -0.7568024992942810058593750
Runtime of SP: 0.0410000000000000172

The double precision floating point function results of 1.000000 is: 0.8414709848078965048756572
The double precision floating point cmath results of 1.000000 is: 0.8414709848078965048756572
The double precision floating point function results of 2.000000 is: 0.9092974268256409642319227
The double precision floating point cmath results of 2.000000 is: 0.9092974268256817094169264
The double precision floating point function results of 3.000000 is: 0.1411200078587149242537180
The double precision floating point cmath results of 3.000000 is: 0.1411200080598672135234750
The double precision floating point function results of 4.000000 is: -0.7568025787396138737150864
The double precision floating point cmath results of 4.000000 is: -0.7568024953079282024503982
Runtime of DP: 0.0429999999999999656
```

We can see that the result of the taylor calculation function I wrote is generally consistent with the system's own cmath function, so there is no principled problem in the result. As the single operation time is too short, I used a large loop to amplify the time comparison and can see that the double precision operation is slower than the single precision operation in non-AVX environment. However, when the input x is too large, say 2000 or 2500, then the single precision based taylor function fails and Inf overflows occur at each iteration step, while the double precision based taylor function does not fail and still performs the calculation normally. This is due

to the fact that they take up different amounts of memory. Single precision floating point numbers take up 4 bytes (32 bits) of memory to store a floating point number, including 1 bit of sign, 8 bits of order and 23 bits of mantissa. Double precision floating point numbers use 8 bytes (64 bits) of memory to store a floating point number, including 1 bit for the sign, 11 bits for the order, and 52 bits for the tail. The range of values for single precision floating point numbers is -3.4e38 to 3.4e38, while the absolute value range of numbers that can be represented by double precision floating point numbers is -2.23e308 ~ 1.79e308.

b) Explore the benefits of compiling on the Discovery cluster with floating point vector extensions (e.g., AVX). Use the single-precision code from part (a). First run on a node on Discovery that does not support AVX-512. Then run on a node that supports AVX-512 and report on the performance benefits. Additional information is provided on AVX support on Discovery.

System: Intel(R) Xeon(R) Platinum 8276 CPU @ 2.20GHz
CPU(s): 56
Thread(s) per core: 1

```
number of terms:
10
The single precision floating point function results of 1.000000 is: 0.84147095680236816406250000
The single precision floating point cmath results of 1.000000 is: 0.84147095680236816406250000
The single precision floating point function results of 2.000000 is: 0.90929740667343139648437500
The single precision floating point cmath results of 2.000000 is: 0.90929740667343139648437500
The single precision floating point function results of 3.000000 is: 0.14112009108066558837890625
The single precision floating point cmath results of 3.000000 is: 0.14112000167369842529296875
The single precision floating point function results of 4.000000 is: -0.75680249929428100585937500
The single precision floating point cmath results of 4.000000 is: -0.75680249929428100585937500
Runtime of SP: 0.08999999999999999667
```

```
number of terms:
10
The single precision floating point function results of 1.000000 is: 0.84147095680236816406250000
The single precision floating point cmath results of 1.000000 is: 0.84147095680236816406250000
The single precision floating point function results of 2.000000 is: 0.90929740667343139648437500
The single precision floating point cmath results of 2.000000 is: 0.90929740667343139648437500
The single precision floating point function results of 3.000000 is: 0.14112009108066558837890625
The single precision floating point cmath results of 3.000000 is: 0.14112000167369842529296875
The single precision floating point function results of 4.000000 is: -0.75680249929428100585937500
The single precision floating point cmath results of 4.000000 is: -0.75680249929428100585937500
Runtime of SP: 0.07000000000000000666
```

The image above shows the results in the non-AVX environment and the image below shows the results in the AVX environment, which shows the speedup in floating point operations in the AVX environment. AVX-512 is a new set of instructions, all vector instructions, which further extends the instruction width to 512 bits, doubling the width and number of data registers and the width of FMA cells compared to AVX2, so that 32 double-precision and 64 single-precision floating-point operations, or 8 64-bit and 16 32-bit integers, can be packed in each

clock cycle.

c) Continuing with part (b), generate an assembly listing (using the -S flag) and identify 2 different AVX instructions that the compiler generated, explaining their operation.

vmovss: move or merge scalar single-precision floating-point value.
vmovd: move doubleword.

d) Provide both IEEE 754 single and double precision representations for the following numbers: 2.1, 6300, and -1.044.

(Hexadecimal based 4-bytes representation for single-precision values and 8-bytes representation for double-precision values)
2.1 (single precision): 40 | 06 | 66 | 66
2.1 (double precision): 40 | 00 | CC | CC | CC | CC | CC | CD
6300 (single precision): 45 | C4 | E0 | 00
6300 (double precision): 40 | B8 | 9C | 00 | 00 | 00 | 00 | 00
-1.044 (single precision): BF | 85 | A1 | CB
-1.044 (double precision): BF | F0 | B4 | 39 | 58 | 10 | 62 | 4E