ME 459 Homework 10

Jackson Fox

Due 12/13/2021

Task 1

Discussion

The maximum int value that I found on my machine was 2147483647, which is $2^{31} - 1$. Once I added one to this, the value turned to -2147483648, or $-1 * 2^{31}$ The reason this occurs is because an int has a size of 4 bytes, which is 32 bits of information. Because each bit can be 1 or 0, this would imply that the maximum value of int should be $2^{32} - 1$. However, the lead bit is reserved as a positive or negative indicator, with 0 meaning positive and 1 meaning negative. This leaves 31 bits to store actual information. Therefore, when adding 1 to 2147483647, it switches the first bit to a 1 which flips the entire value to negative. For the same reason, the most negative value that I could get for int on my machine was -2147483648, and when I subtracted one from this the result was 2147483647. An interesting thing to note is that the binary value used for -2147483648 is actually 10000000000000000000000000000000, which is actually the number . This is counter intuitive, because the system is seeing the lead bit as the negative indicator, which would imply the remaining values would be 0. However, the way that negative values are read is that the system sees the 1 in the 32 spot as meaning $-1*2^{31}$, and then adds the value of whatever bits are to the right of that to bring towards 0. Therefore, 11111111111111111111111111111111 represents $-1 * 2^{31} + 2^{31} - 1$, or -1.

Task 2

Part a - Timing Values

Six executions of the task2.c script were recorded, the individual and average execution times in milliseconds for the three functions are listed below:

	Average	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6
Sum	**1244.097**	1243.987	1244.063	1242.253	1243.630	1244.013	1246.637
SumSines	**14872.291**	15019.113	14906.154	14909.59	14900.651	14962.891	14535.345
Fusion	**15287.365**	15091.541	15018.344	15018.804	15026.920	16763.415	14805.463

Part b - Discussion

The average execution of the SumSines function was significantly slower than the Sum function, taking nearly 12 times as long. This is due to the fact that each iteration of the for loop in the SumSines function is required to not only retrieve the arr values and sum, but also call the separate "sin" function which take significant time. The Sum function only has to retrieve the values and perform a simple addition, which can leverage the cache. Fusion completes both tasks, and was consequently the slowest.

However, the increase in completion time from SumSines to Fusion was about half the time it took to complete Sum, as opposed to simply a combination of the times for both. This is due to the fact that both Sum and SumSines have to get the next value of arr[ ] during each iteration of the loop, and in the Fusion function that is only done once. For each iteration of Sum that likely took half the total time, with the other half going to the arithmetic. Therefore, when the functionalities were combined in the same loop the only addition to the time taken in SumSines is the addition of the plain addition step (half of the Sum execution time) as the next value for arr[ ] is already read and does not need to be read twice.

Task 3

Mathematical Reasoning

L'Hopitals rule states that for functions $f(x)$ and $g(x)$ which are differentiable on open interval I containing the point c, where $g'(x)$ is not equal to 0 (besides potentially at c), and

$$\lim_{x \to c} \frac{f'(x)}{g'(x)} = L$$

if

$$\lim_{x \to c} f(x) = \lim_{x \to c} g(x) = 0$$

or

$$\lim_{x \to c} |f(x)| = \lim_{x \to c} |g(x)| = \infty$$

then

$$\lim_{x \to c} \frac{f(x)}{g(x)} = L$$

Take the expression

$$\lim_{x \to c} \frac{sin(x)}{x}$$

where $f(x) = sin(x)$, $g(x) = x$ and c = 0. For the open interval I (all real numbers) containing c, both $f(x)$ and $g(x)$ are differentiable:

$$f'(x) = cos(x)$$

$$g'(x) = 1$$

Additionally:

$$\lim_{x \to 0} f(x) = sin(0) = 0$$

$$\lim_{x \to 0} g(x) = 0$$

and

$$\lim_{x \to 0} \frac{f'(x)}{g'(x)} = \lim_{x \to 0} \frac{cos(x)}{1} = 1 = L$$

. Therefore, the conditions for L'Hopitals rule are satisfied and:

$$\lim_{x \to 0} \frac{sin(x)}{x} = \lim_{x \to 0} \frac{cos(x)}{1} = 1$$

Part c and d - Discussion

xf was found to be 0.00055894261458888649940490722656250 0 whereas xd was found to be 0.0000000214911917503701875615395336 34. Xf was significantly higher than Xd - this is because float is only a 32 bit precise value whereas double is 64 bit - hence when using doubles, the system could track to values much closer to 1 than when using floats.