CSC241: Assignment 0x14, SIMD

The textbook gives two SIMD implementations of a sine function, in both single precision and double precision, on pages 394-396. In this assignment, you'll write an alternative implementation and a slight adaptation of the book's version, so we can compare them.

Taylor Series for Sine

The book's implementation uses a common technique for approximating functions by polynomials, namely Taylor series. We have from calculus that

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Of course we can't use infinitely many terms, so we stop the series at some finite number of terms. The error is no larger than the first omitted term. So for example, the book's implementation goes through the exponent 15, for a total of eight terms:

$$\begin{aligned}\sin x &\approx \sum_{n=0}^{7} \frac{(-1)^n x^{2n+1}}{(2n+1)!} \\ &\approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!} - \frac{x^{15}}{15!}.\end{aligned}$$

The error is therefore no worse than the first omitted term, i.e.

$$|E| < \frac{x^{17}}{17!}.$$

If we restrict our attention to values of x in the range of, say, $[-\pi/2, \pi, 2]$, then the maximum error is

$$|E_{max}| = (\pi/2)^{17}/17! \approx 6.06694 \times 10^{-12}$$

Minor Modification

We will make a minor modification to our function, in order to set ourselves up for the alternative implementations and allow for some apples-to-apples comparisons. Instead of writing a function to evaluate $\sin x$, we will instead write a function to evaluate $\sin\left(\frac{\pi x}{2}\right)$, where $y = 2x$. Hence our input will be on the interval $[-1, 1]$. This means we can think of our inputs as being sort of degrees: $x = 1$ corresponds to $90°$, etc. This change, however, will require us to re-compute our Taylor series coefficients: instead of $1/n!$, we will now have $((\pi/2)^n)/n!$. This makes our polynomial

$$\begin{aligned}\sin\left(\frac{\pi}{2}x\right) \approx &-\frac{\pi^{15}x^{15}}{42849873690624000} + \frac{\pi^{13}x^{13}}{51011754393600} - \frac{\pi^{11}x^{11}}{81749606400} \\ &+\frac{\pi^9 x^9}{185794560} - \frac{\pi^7 x^7}{645120} + \frac{\pi^5 x^5}{3840} - \frac{\pi^3 x^3}{48} + \frac{\pi x}{2}\end{aligned}$$

Chebyshev Approximation

Chebyshev Polynomials give us a different way to attack this problem. The Chebyshev polynomials are defined recursively, with $T_0 = 1, T_1 = x$, and $T_{n+1} = 2xT_n - T_{n-1}$. We wish to find coefficients b_k and c_k so that we can approximate a function as

$$f(x) \approx \sum_{k=0}^{n} b_k T_k(x) \approx \sum_{k=0}^{n} c_k x^k$$

for some value of n. Finding these coefficients is a bit complicated and took me a while to do in *Mathematica*. Unlike Taylor series, though, the coefficients depend on n also, so that you can't just tack on more terms without having to recompute all the coefficients. However, the coefficients are close, so you wouldn't necessarily notice a difference without looking at them closely.

Apollo Guidance Computer

The motivation for this problem stems from the source code from the Apollo Guidance Computer. It approximated the sine of an angle using

$$\sin\left(\frac{\pi}{2}x\right) \approx 1.5706268x - 0.6432292x^3 + 0.0727102x^5,$$

which was published in Hastings et al., *Approximations for Digital Computers*, 1955. I used Mathematica to compute a similar approximation:

$$\sin x \approx 0.0729346x^5 - 0.643458x^3 + 1.57066x,$$

which was more accurate by a bit for $x \in [-.6, .6]$ or so, but slightly less accurate outside of that interval. I'm not quite sure why I my numbers are slightly off of Hastings'.

The arithmetic on the AGC is hard to compare to modern computers, but it stored numbers using anywhere from single precision (14 bits plus sign and parity) to triple precision (14 + 2*15 bits plus 1 sign bit and 3 parity bits, I think). Since ARM floating point can at minimum deal with 32 bit floats, we'll pretend that we can make an apples-to-apples comparison with 32 bits floats or 64 bit doubles.

The Assignment

For this assignment, you will write a few functions:

- The fifteenth degree Taylor approximation, in 64 bit, using SIMD. For this function you can essentially steal the book's version, and replace its coefficients with the modified ones (since the book does $\sin x$ and we're doing $\sin(\pi x/2)$).
- The Apollo fifth degree polynomial approximation, in 64 bit, using SIMD. The coefficients listed below are the actual Apollo coefficients, not my version.

- The fifteenth degree Chebyshev approximation, in 64 bit, using SIMD. These coefficients I computed.

The user will input an angle y in degrees, and the program will compute $x = y/90$ and then $\sin\left(\frac{\pi}{2}x\right)$. The program should then output the approximated values. Use strings to make it clear which value is which.

Coefficients

Since the sine function is odd, we only actually have odd powers of x in our polynomials. We need to compute/convert the coefficients and list them in float or double format. I wrote a script to do this quickly with Mathematica. I'll use the notation c_i as the coefficient of the x^i term. Assembly-ready form: download it here.

Apollo Coefficients

Coefficient	Decimal	Float	Double
c_1	1.5706268	`0x3fc90a4d`	`0x3ff92149914389c3`
c_3	-0.643229	`0xbf24aaab`	`0xbfe49555673aa1bc`
c_5	0.0727102	`0x3d94e916`	`0x3fb29d22bb15eb24`

Chebyshev Coefficients

Coefficient	Decimal	Float	Double
c_1	1.5707963267948964	`0x3fc90fdb`	`0x3ff921fb54442d17`
c_3	-0.64596409750622323	`0xbf255de7`	`0xbfe4abbce625bd83`
c_5	0.079692626245683363	`0x3da335e3`	`0x3fb466bc677522bd`
c_7	-0.0046817541314470625	`0xbb996966`	`0xbf732d2cce1ea145`
c_9	$0.00160441169563106 30$	`0x39283c19`	`0x3f25078327046959`
c_{11}	$-3.5988107063136795 * 10^{-6}$	`0xb6718319`	`0xbece30631bdf732d`
c_{13}	$5.6883180640177953 * 10^{-8}$	`0x33744fb8`	`0x3e6e89f6fe44fe7b`
c_{15}	$-6.4494712378091340 * 10^{-10}$	`0xb031481f`	`0xbe062903d02bb153`

Taylor Coefficients

Coefficient	Decimal	Float	Double
c_1	1.5707963267948966	`0x3fc90fdb`	`0x3ff921fb54442d18`
c_3	-0.64596409750624625	`0xbf255de7`	`0xbfe4abbce625be53`
c_5	0.079692626246167045	`0x3da335e3`	`0x3fb466bc6775aae2`
c_7	-0.0046817541353186881	`0xbb996966`	`0xbf732d2cce62bd86`
c_9	$0.00160441184787359 82$	`0x39283c1a`	`0x3f250783487ee782`
c_{11}	$-3.5988432352120853 * 10^{-6}$	`0xb67183a8`	`0xbece3074fde8871f`
c_{13}	$5.6921729219679268 * 10^{-8}$	`0x33747a1a`	`0x3e6e8f434d018d63`
c_{15}	$-6.6880351098114672 * 10^{-10}$	`0xb037d6dd`	`0xbe06fadb9f155744`