Next Silicon: CM Home Assignment

Dorde Zivanovic
LinScale

April 24, 2025

Contents

1 Introduction

This report contains the responses to the tasks stated in the home project pdf. It is accompanied with the repository that contains reproducible solutions with the installation instructions alongside: experiments and tests according to the task requirements.

2 Code Analysis and Documentation

In this section we analyze the existing code shown in Algorithm 1.

2.1 The Existing Implementation: Code Drawbacks

The code is written for C and follows the following bad practices:

1. Reusing the variable multiple times, `(float)M_PI` and `2.0f * (float)M_PI`, (lines 3, 4, 6, 6, 5, 7 of Algorithm 1);

2. Not using auto in order to automatically deduce types since the results of all the statements are known;

3. Cleaning if loop to be more understandable: `fmodf` returns the result in the range $(-2\pi, 2\pi)$. Then, now it is obvious that one checks whether the number is outside of the range $[-\pi, \pi]$, and then updates `x` for 2π period, so the method works from the number in the range $[-\pi, \pi]$;

4. Adding more verbosity ();

5. Reusing variable names -> more verbose names should be used in order to improve readability of the code. The c ompiler will optimize for the least number of variables/registers to be used;

6. Renaming function names and migrating these functions to the corresponding headers and sources that would contain the custom maths functions.

2.2 Drawbacks: numerical and implementation issues

Here, I will give state several main drawbacks in terms of the implementation and numerical accuracy. The division by In the next subsection, I will list the drawbacks related to the method itself.

2.3 Mathematical Analysis

2.4 Failures

2.5 Test Plan

3 Conclusion

Summarize your findings or thoughts here. [1]

1: **Input:** A float (IEEE-754) number
2: **Output:** A float (IEEE-754) sine value of this number computed using Taylor Series.
3: **Steps:**

```
float fp32_custom_sine(float x)
{
    x = fmodf(x, 2.0f * (float)M_PI);
    if (x > (float)M_PI)
        x -= 2.0f * (float)M_PI;
    else if (x < -(float)M_PI)
        x += 2.0f * (float)M_PI;
    float result = 0.0f;
    float term = x;
    float x_squared = x * x;
    int sign = 1;
    for (int n = 1; n <= 7; n += 2)
    {
        result += sign * term;
        sign = -sign;
        term = term * x_squared;
        term = term / (float)(n + 1);
        term = term / (float)(n + 2);
    }
    return result;
}
```

Algorithm 1: Algorithm with Code Listing

References

[1] Tom M Apostol. Mathematical analysis. Narosa Publishing House Pvt. Ltd., 1985.