CENG314 **Spring 2020**

ASSIGNMENT 3

Due date: 24.06.2020 23:00

The purpose of this assignment is to utilize optimization techniques to get faster program executions.

Suppose we have a function to evaluate a polynomial, where a polynomial of degree n is defined to have a set of coefficients a_0, a_1, a_2, ..., a_n . For a value x, we evaluate the polynomial by computing:

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

This evaluation can be implemented by the following function, having as arguments an array of coefficients *a*, a value *x*, and the polynomial degree, *degree*. In this function, we compute both the successive terms of the equation and the successive powers of x within a single loop:

```
double poly(double a[], double x, int degree)
{
      long int i;
      double result = a[0];
      double xpwr = x;
      for (i = 1; i <= degree; i++) {
            result += a[i] * xpwr;
            xpwr = x * xpwr;
      }
      return result;
}
```

We can reduce the number of multiplications in evaluating a polynomial by applying Horner's method. The idea is to repeatedly factor out the powers of x to get the following evaluation:

$$a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n) \cdots))$$

Using Horner's method, we can implement polynomial evaluation using the following code:

```
double horner_poly(double a[], double x, int degree)
{
      long int i;
      double result = a[degree];
      for (i = degree-1; i >= 0; i--)
            result = a[i] + x*result;
      return result;
}
```

(For further details about the functions and their expected performance, you can see the Practice Problems 5.5 and 5.6 (and Solutions) in the given reference chapter as part of the lecture. However, in the scope of this assignment, we do not focus on the performance differences of those two versions.)

In this assignment, you are required to write faster polynomial evaluation using the optimization techniques we have seen in the lecture. Firstly, you will apply optimization techniques on `poly` and `horner_poly` to explore possible performance improvements. Finally, you are required to write an

optimized version for mixing of them (Hint: Consider the version $a_0 + x *(a_1 + a_2 * x) + x^3 * (a_3 + a_4 * x) + x^5 * (a_5 + a_6 * x) + ...$), and optimize it as much as possible.

In order to get more significant results, it is better to execute a code with many calls to the functions with large degree values (as given in the example source file).

You will write your optimized code, execute original and optimized versions (with gcc compiler, no optimization flag), collect execution times and write a report including your results and discussion/comments. In your discussion, you should include/explain all program versions and results used in intermediate steps.

Bonus (up to 50 points):
You are required to change the optimization levels (https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html) provided by gcc, which are -O1, -O2, -O3, -Ofast. After compiling the original and optimized functions with each level, you will execute the programs and collect the execution times. You will include your results and comments about the effect of compiler optimizations on each version.

It is highly recommended (but not required) that you prepare a script to be able to automate your experiments.

For both parts, you need to execute each version/experiment for 10 times, and report the average of the values in your tables.

Notes:

- You are required to submit your source file(s) and a report that includes your code with explanations, your results with tables and comments about the results.
- You need to work individually, no group work is allowed.
- No late homework will be accepted.

Submission: You are required to submit your report and codes to CMS as a compressed file named as yourstudentnumber_A3.tar.gz (e.g. If your student number is 201812345678, the file name must be 201812345678_A3.tar.gz).