Laboratorium 1 – Arytmetyka komputerowa

Tomasz Belczyk
20.03.2021

Metody Obliczeniowe w Nauce i Technice
Informatyka niestacjonarna 2020/2021
Wydział Informatyki, Elektroniki i Telekomunikacji
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

1. Treść zadań

1. Napisać program liczący kolejne wyrazy ciągu:

$$x(n+1) = x(n) + 3.0 * x(n) * (1 – x(n))$$

Startując z punktu x(0) = 0.01. Wykonać to zadanie dla różnych reprezentacji liczb (float, double). Dlaczego wyniki się rozbiegają?
Uwaga: Należy wprowadzić zmienne pomocnicze, aby uniknąć obliczeń w rejestrach procesora
2. Napisać program liczący ciąg z wcześniejszego zadania, ale wg wzoru

$$x(n+1) = 4.0 * x(n) – 3.0 * x(n) * x(n)$$

Porównać z wynikami z wcześniejszego zadania.
3. Znaleźć „maszynowe epsilon” czyli najmniejszą liczbę a, taką że a+1>1

2. Podejście do rozwiązania zadań

Rozwiązując zadania wykorzystamy narzędzie VS Code jak i obraz docker build’a uruchomiony na dockerze pozwalający na kompilowanie, uruchamianie programów w ubuntu. Rozwiązując zadania wykorzystamy dwa typy danych: double i float

Zadanie 1

Implementujemy algorytm do liczenia ciągu z zadania dla obu typów danych. Prezentuje się on następująco

```
30  double f_double(double x) {
31    double f_n;
32    if(x == 0.00) {
33      return 0.01;
34    }
35
36    f_n = f_double((double)x - 1);
37
38    return f_n + 3.0 * f_n * (1.0 - f_n);
39  }
40
41  float f_float(float x) {
42    float f_n;
43    if(x == 0) {
44      return 0.01f;
45    }
46
47    f_n = (float)f_float(x-1);
48
49    return (float)(f_n + 3.0f * f_n * (1 - f_n));
50  }
```

Metoda f_double to algorytm rekursywny liczący wyrazy ciągu dla double. Odpowiednio f_float liczy wyrazy ciągu dla float. Całość programu wygląda tak

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

double f_double(double x);
float f_float(float x);

int main (void)
{
  double sum_double = 0;
  float sum_float = 0;

  for(int i = 0; i < 10000; i++) {

    sum_double += f_double((double)i);

  }
  printf("double %f\n", sum_double);


  for(int i = 0; i < 10000; i++) {
    sum_float += f_float((float)i);
  }

  printf("float: %.12f\n", sum_float);

  return 0;
}

double f_double(double x) {
  double f_n;
  if(x == 0.00) {
    return 0.01;
  }

  f_n = f_double((double)x - 1);

  return f_n + 3.0 * f_n * (1.0 - f_n);
}

float f_float(float x) {
  float f_n;
  if(x == 0) {
    return 0.01f;
  }

  f_n = (float)f_float(x-1);

  return (float)(f_n + 3.0f * f_n * (1 - f_n));
}
```

Wyniki działania programu przedstawiono w tabeli 1. Wyniki znacząco różnią się od siebie co jest zamierzonym rezultatem ze względu na standard reprezentacji binarnej i operacji na liczbach zmiennoprzecinkowych (IEEE 754). Liczby pojedynczej precyzji, tj. float, posiadają zakres od od około $\pm 1.18 \cdot 10^{-38}$ do około $\pm 3.4 \cdot 10^{38}$. W czasach, gdy popularne komputery nie miały koprocesorów matematycznych koszt prowadzenia obliczeń w pojedynczej precyzji był znacząco niższy od kosztu

obliczeń w podwójnej precyzji, więc mimo oczywistych niedostatków powszechnie korzystano z tych pierwszych. Obecnie jednak ta różnica nie jest aż tak znacząca i obliczenia w podwójnej precyzji są znacznie częściej stosowane. Z kolei liczby podwójnej precyzji ma zakres od około $\pm 2.2 \cdot 10^{-308}$ do około $\pm 1.8 \cdot 10^{308}$ co daje o wiele większe możliwości i dokładność wyliczeń. Typ double(podwójna precyzja) ma o wiele większą precyzje co z kolei wpływa na większe zużycie zasobów, lecz w dzisiejszych czasach gdzie standardem są wielo rdzeniowe procesory o dużej mocy nawet w laptopach/komputerach z słabymi podzespołami nie ma to znaczącego wpływu. Typ float jest natywnie wspierany przez koprocesory. Mało tego, tam gdzie priorytetem jest największa wydajność, używa się instrukcji SSE procesora, operujących właśnie na zmiennych typu float. Z wyżej wymienionych powodów wyniki są rozbieżne.

Zadanie 2

Modyfikujemy algorytmy w następujący sposób:

```
30  double f_double(double x) {
31    double f_n;
32    if(x == 0.00) {
33      return 0.01;
34    }
35
36    f_n = f_double((double)x - 1);
37
38    return 4.0 * f_n - 3.0 * f_n * f_n;
39  }
40
41  float f_float(float x) {
42    float f_n;
43    if(x == 0) {
44      return 0.01f;
45    }
46
47    f_n = (float)f_float(x-1);
48
49    return 4.0f * f_n - 3.0f * f_n * f_n;
50  }
```

Reszta programu pozostaje niezmienna. Wyniki działania programu przedstawiono w tabeli 2. Biorąc pod uwagę wyniki z zadania 1 dalej widzimy że wyniki się różnią. Lecz jest to mniejsza różnica niż w przypadku programu z zadania 1 ponieważ algorytm ciągu jest dokładniejszy.

Zadanie 3

Modyfikujemy program z zadania 2:

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

double f_double(double x);
float f_float(float x);

int main (void)
{
  double min_double,double_result,sum_double = 0;
  float min_float,float_result,sum_float = 0;

  for(int i = 0; i < 10000; i++) {
     double_result = f_double((double)i);
     if(i == 0) {
         min_double = double_result;
     }
     else if(double_result + 1 > 1 && double_result + 1 < min_double) {
         min_double = double_result;
     }
    sum_double += double_result;
  }
  printf("double %f min: %f\n", sum_double, min_double);


  for(int i = 0; i < 10000; i++) {
    float_result = f_float((float)i);
    if(i == 0) {
         min_float = float_result;
     }
    else if(float_result + 1.0f > 1.0f && float_result < min_float) {
        min_float = float_result;
    }
    sum_float += float_result;
  }

  printf("float: %.12f min: %.12f\n", sum_float, min_float);

  return 0;
}
```

Dla typu double najmniejsze a takie że a + 1 > 1 wynosi 0.010000

Dla typu float najmniejsze a takie że a + 1 > 1 wynosi 0.000020

3. Tabele, wyniki

Tabela 1

LP	Float	Double	Różnica
1	6830.709472	6618.791067	211.918405

Tabela 2

LP	Float	Double	Różnica
1	6762.753417	6681.738683	81.014734

4. Wnioski

Typu float możemy używać np. przy obliczeniach związanych z grafiką gdzie nie używamy dużych wartości i nie operujemy na dużych liczbach tylko na liczbach do kilku miejsc po przecinku, które mogą być obarczone niedużym błędem. Typu float możemy używać do przechowywania wartości temperatury, wilgotności itp. o ile nie są to sprzęty dużej dokładności. Z kolei typu double należy używać wszędzie tam gdzie zależy nam na większej dokładności w stosunku do float i chcemy zmniejszyć ryzyko wystąpienia błędów. Jeśli chcemy używać dużych wartości liczbowych z wartością ułamkową także możemy użyć double jednakże trzeba pamiętać o ograniczeniach.