https://github.com/krzsztfwtk

https://github.com/krzsztfwtk/polsl-ja-proj

Prezentacja projektu z Języków Asemblerowych

„Obliczanie wartości wielomianu za pomocą algorytmu Hornera”

krzsztfwtk

polsl-ja-proj

Gliwice 2025

https://github.com/krzsztfwtk/polsl-ja-proj

1 Wprowadzenie

Wielomiany są podstawowymi funkcjami w matematyce i informatyce, wykorzystywanymi w różnych dziedzinach, takich jak analiza numeryczna, przetwarzanie sygnałów czy grafika komputerowa. Obliczanie wartości wielomianu dla danego argumentu jest kluczowe w wielu algorytmach i aplikacjach.

1.1 Obliczanie Wartości Wielomianu

Obliczanie wartości wielomianu polega na wyznaczeniu wyniku funkcji wielomianowej $W(x)$ dla danej wartości x. Wielomian stopnia n można zapisać jako:

$$W(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

gdzie a_i to współczynniki wielomianu.

1.2 Metoda Podstawowa (Nieefektywna)

Podstawowa metoda polega na bezpośrednim obliczeniu każdego wyrazu wielomianu i zsumowaniu wyników.

$$W(x) = \sum_{i=0}^{n} a_i x^i$$

```
procedure MetodaPodstawowa(a, x, n)
begin
    suma := 0;
    for i := 0 to n do
    begin
        potega_x := 1;
        for j := 1 to i do
        begin
            potega_x := potega_x * x;
        end;
        wyraz := a[i] * potega_x;
        suma := suma + wyraz;
    end;
    return suma;
end;
```

Lit. 1 Pseudokod algorytmu obliczającego wartość wielomianu metodą podstawową

Złożoność obliczeniowa przy założeniu, że operacją dominującą jest mnożenie:

$$T = \sum_{i=0}^{n} n = \frac{1}{2}n^2 = O(n^2)$$

gdzie n to ilość współczynników wielomianu.

Duża liczba operacji mnożenia, która wynika z potęgowania sprawia, że algorytm ten jest nieefektywny.

2 Metoda Hornera (Optymalna)

Metoda Hornera jest bardziej efektywna od metody podstawowej, ponieważ przekształca wielomian do postaci zagnieżdżonej, redukując liczbę operacji mnożenia:

$$W(x) = a_0 + x\left(a_1 + x\left(a_2 + x(\dots + x(a_{n-1} + xa_n))\right)\right)$$

```
procedure Horner(a, x, n)
begin
    wynik := a[n];
    for i := n - 1 downto 0 do
    begin
        wynik := wynik * x + a[i];
    end;
    return wynik;
end;
```

Lst. 2 Pseudokod algorytmu obliczającego wartość wielomianu metodą Hornera

Złożoność obliczeniowa przy założeniu, że operacją dominującą jest mnożenie:

$$T = \sum_{i=0}^{n} 1 = n = O(n)$$

2.1 Wykorzystanie wielowątkowości

Równoległe wykonywanie obliczeń poprzez podział zadań między wiele wątków procesora pozwala obliczać wartość wielomiany we wielu punktach (dla wielu x) jednocześnie. Zbiór punktów x dzielony jest na części przypisane do poszczególnych wątków. Każdy wątek oblicza wartości wielomianu dla swojej części danych niezależnie.

```
procedure HornerWielowatkowo(a, x_tablica, n, liczba_watkow)
begin
    podziel x_tablica na liczba_watkow części;
    parallel for każda_część in części do
    begin
        for x in każda_część do
        begin
            wynik := Horner(a, x, n);
            zapisz wynik;
        end;
    end parallel for;
end;
```

Lst. 3 Pseudokod algorytmu obliczającego wartości wielomianu metodą Hornera z wykorzystaniem wielowątkowości

Na procesorze Intel Core i7-11700K, który posiada 8 rdzeni i 16 wątków (dzięki Hyper-Threading) oczekiwane przyspieszenie algorytmu dzięki zastosowaniu wielowątkowości wynosi 16 razy w porównaniu z wersją jednowątkową. Może okazać się, że będzie to mniej ze względu na narzut związany z zarządzaniem wątkami.

https://github.com/krzsztfwtk/polsl-ja-proj

2.2 Wykorzystanie instrukcji wektorowych

Instrukcje wektorowe np. AVX (Advanced Vector Extensions) pozwalają na jednoczesne wykonywanie operacji na wielu rejestrach (SIMD). Pozwala to obliczanie wartości wielomianu w **wielu punktach jednocześnie na jednym wątku procesora**.

```
procedure HornerWektorowo(a, x_wektor, n)
begin
    wynik_wektor := zerowy_wektor();

    for i := n - 1 downto 0 do
    begin
        wynik_wektor := wynik_wektor * x_wektor + wektor(a[i]);
    end;
    return wynik_wektor;
end;
```

Lst. 4 Pseudokod algorytmu obliczającego wartości wielomianu z wykorzystaniem instrukcji wektorowych

Rejestry AVX pozwalają na przetwarzanie 8 wartości typu float jednocześnie. Zatem oczekiwane przyspieszenie programu wykorzystującego instrukcje AVX wynosi 8 razy w porównaniu z programem wykonującym operacji skalarnych.

2.3 Jednoczesne wykorzystanie wielowątkowości i instrukcji wektorowych

Połączenie wielowątkowości i instrukcji AVX pozwala na znaczące przyspieszenie algorytmu na nowoczesnych procesorach. Łączne wykorzystanie obu technik skutkuje multiplikatywnym przyspieszeniem, ponieważ każda z nich działa na innym poziomie: wielowątkowość na poziomie wątków, a SIMD na poziomie instrukcji.

```
procedure HornerWielowatkowoWektorowo(a, x_tablica, n, liczba_watkow)
begin
    podziel x_tablica na liczba_watkow części;
    parallel for t := 0 to liczba_watkow - 1 do
    begin
        start := t * rozmiar_części;
        koniec := (t + 1) * rozmiar_części;
        for i := start to koniec - 8 step 8 do
        begin
            x_wektor := załaduj_wektor(x_tablica[i]);
            wynik_wektor := zerowy_wektor();
            for j := n - 1 downto 0 do
            begin
              wynik_wektor := wynik_wektor * x_wektor + wektor(a[j]);
            end;
            zapisz_wektor(wynik_wektor, wyniki[i]);
        end;
        for i := koniec - (koniec mod 8) to koniec do
        begin
            wynik := Horner(a, x_tablica[i], n);
            wyniki[i] := wynik;
        end;
    end parallel for; end;
```

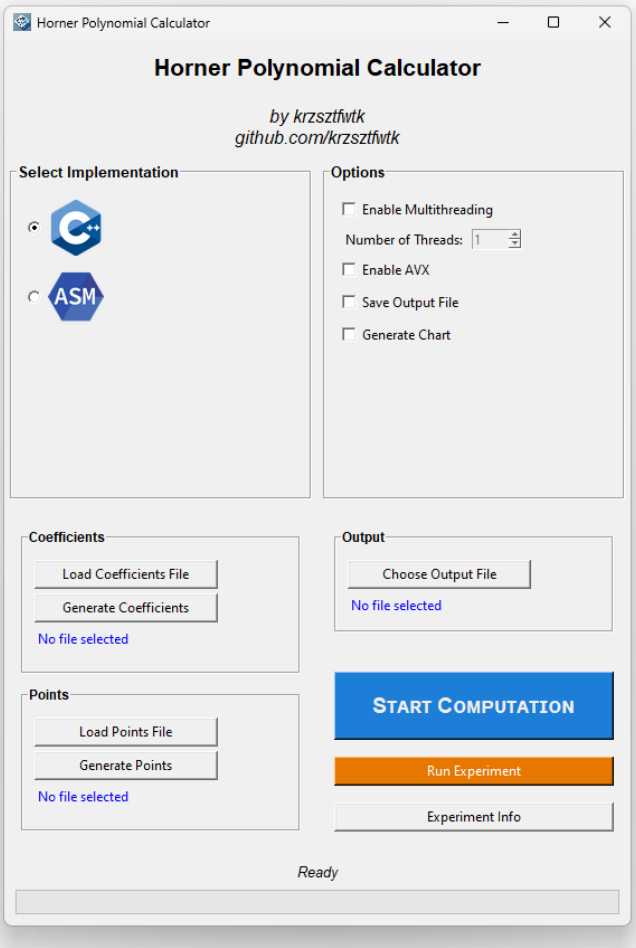
Lst. 5 Pseudokod algorytmu obliczającego wartości wielomianu z wykorzystaniem jednocześnie wielowątkowości oraz instrukcji wektorowych

https://github.com/krzsztfwtk/polsl-ja-proj

Na procesorze, który posiada 16 wątków oraz obsługuje instrukcje AVX umożliwiające przetwarzanie 8 wartości typu float jednocześnie, oczekiwane przyspieszenie wynosi $16 \cdot 8 = 128$ razy w porównaniu z wersją jednowątkową bez wektoryzacji. Rzeczywiste przyspieszenie może być mniejsze ze względu na narzut związany z zarządzaniem wątkami oraz ograniczenia przepustowości pamięci.

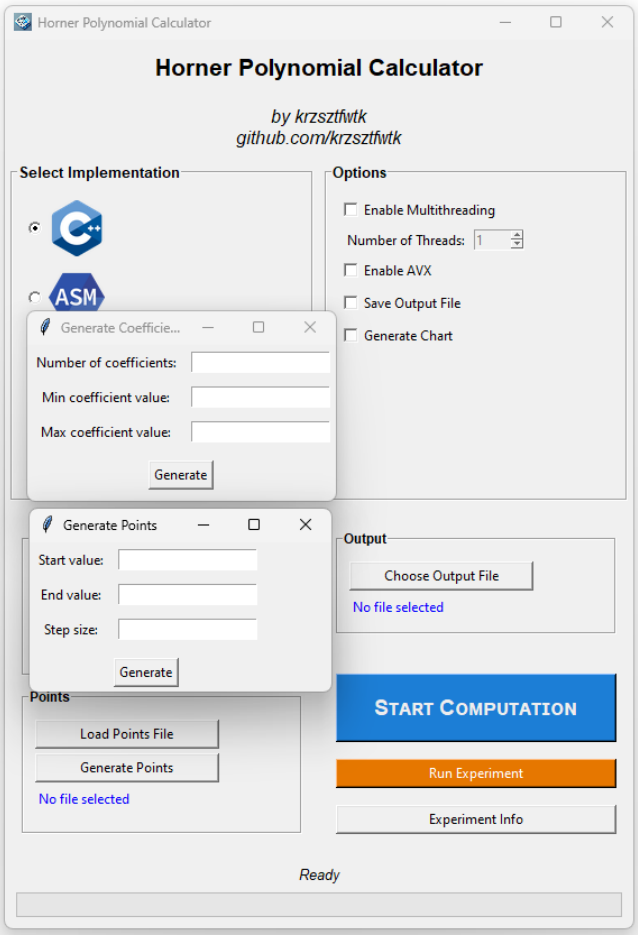
3 Wygląd interfejsu użytkownika

Interfejs graficzny napisany w języku Python umożliwi użytkownikowi łatwe zarządzanie procesem obliczania wartości wielomianu. Główne okno aplikacji składa się z kilku sekcji, które pozwalają na wprowadzenie parametrów, wybór opcji oraz kontrolę przebiegu obliczeń.



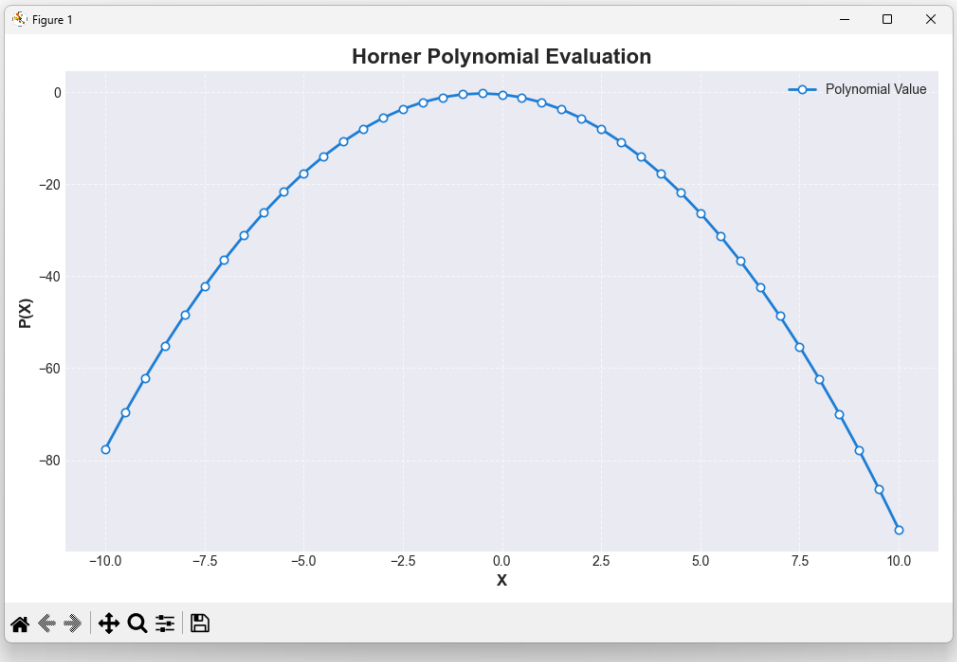
W górnej części interfejsu znajdują się przyciski umożliwiające załadowanie plików z współczynnikami wielomianu oraz punktami, w których wartości mają zostać obliczone. Użytkownik ma również możliwość automatycznego wygenerowania losowych współczynników oraz dużego zbioru punktów za pomocą dedykowanych przycisków. Po wybraniu lub wygenerowaniu plików, nazwy wybranych plików są wyświetlane obok odpowiednich przycisków, co ułatwia orientację.

https://github.com/krzsztfwtk/polsl-ja-proj



Środkowa część interfejsu zawiera opcje wyboru implementacji algorytmu oraz dodatkowych ustawień. Użytkownik może wybrać między implementacją w języku C++ lub asemblerze. Dodatkowo dostępne są pola wyboru (checkboxy) pozwalające na włączenie lub wyłączenie wielowątkowości oraz instrukcji AVX, co daje kontrolę nad wydajnością i sposobem wykonania obliczeń. Istnieje także opcja decydująca o tym, czy po zakończeniu obliczeń ma zostać wygenerowany wykres przedstawiający wyniki. Wykres jest generowany za pomocą biblioteki *matplotlib* w języku Python.

https://github.com/krzsztfwtk/polsl-ja-proj



W dolnej części interfejsu znajduje się przycisk *Start Computation*, który inicjuje proces przetwarzania danych zgodnie z wybranymi ustawieniami. Podczas wykonywania obliczeń wyświetlany jest pasek postępu oraz komunikat informujący o aktualnym statusie operacji. Po zakończeniu obliczeń aplikacja wyświetla czas wykonania oraz, jeśli taka opcja została wybrana, prezentuje wykres wyników.

Interfejs umożliwia użytkownikowi ponowne wykonanie obliczeń z innymi parametrami bez konieczności ponownego uruchamiania aplikacji. Dzięki temu użytkownik może eksperymentować z różnymi ustawieniami i obserwować wpływ zmian na czas wykonania i wyniki obliczeń.

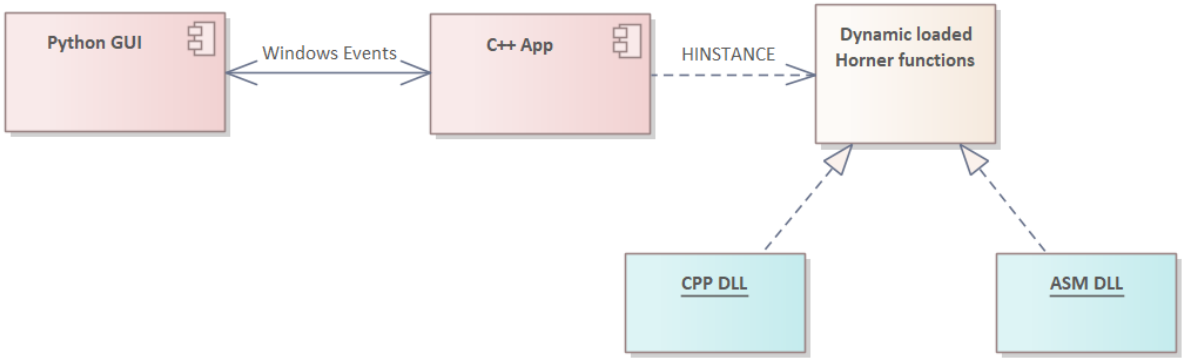
4 Interakcja między GUI a modułem C++

Aplikacja składa się z dwóch niezależnych programów: interfejsu użytkownika napisanego w języku Python oraz modułu obliczeniowego w języku `C++`. Interfejs użytkownika zapewnia graficzne środowisko do obsługi aplikacji, natomiast moduł `C++` odpowiada za wykonywanie intensywnych obliczeń związanych z obliczaniem wielomianu.

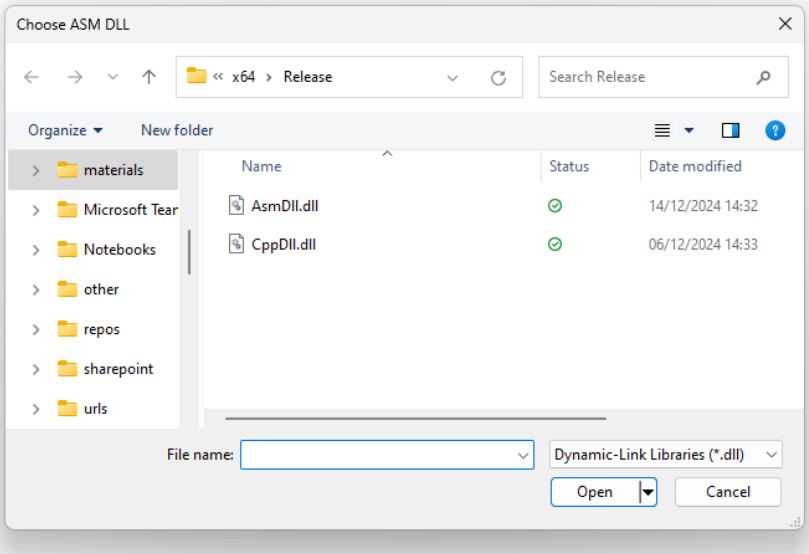
Komunikacja między obiema aplikacjami odbywa się za pośrednictwem sygnałów systemu Windows oraz plików tymczasowych. Po wprowadzeniu przez użytkownika odpowiednich parametrów i ustawień w interfejsie graficznym, aplikacja Python zapisuje dane wejściowe (współczynniki wielomianu i punkty do obliczeń) do plików tymczasowych w katalogu temp. Następnie sygnalizuje modułowi `C++`, że dane są gotowe do przetworzenia. Sygnalizacja odbywa się poprzez zdarzenie systemowe o nazwie *Global\ComputeEvent*.

Moduł `C++` nasłuchuje na to zdarzenie, a gdy zostanie ono ustawione, rozpoczyna proces obliczeń, odczytując dane z plików wejściowych. Po zakończeniu obliczeń wyniki są zapisywane do plików wyjściowych.

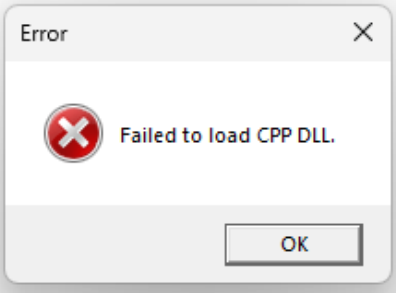
https://github.com/krzsztfwtk/polsl-ja-proj



Taka konfiguracja pozwana na uniknięcie problemów z wielowątkowością, które mogłyby się pojawić, gdyby biblioteki DLL były ładowane przez kod napisany Pythonie (jeżeli interpreter Pythona działałby w jednym wątku, to wielowątkowość implementacji asm86 i C++ nie mogłaby poprawić wydajności).



Implementacja funkcji w asemblerze jest ładowana przez program napisany w C++ jako biblioteka DLL. Analogicznie na najważniejsze funkcje do obliczania wielomianu napisane w C++ również będą ładowane do programu jako biblioteka DLL, co pozwoli na wiarygodne porównanie.



Załadowanie dynamicznych bibliotek w trakcie działania programu będzie możliwe za pomocą funkcji WinAPI takich jak *MessageBox*. Ewentualne błędy zostaną również wyświetlone z użyciem WinAPI. Jeżeli użytkownik poproszony o załadowanie implementacji CPP DLL wskaże na plik DLL zawierający implementację ASM DLL i odwrotnie, to program nie będzie w stanie wykryć odwróconej kolejności. Użytkownik może załadować do programu dowolną implementację funkcji w bibliotece dynamicznej niezależnie od języka programowa, w której została napisana.