

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

Program wielowątkowy z zastosowaniem GitHub Copilot

Autor:
Kamil Gruca

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań	3
2. Analiza problemu	4
2.1. Ogólny problem	4
2.2. Metoda prostokątów	4
2.3. Zrównoleglenie obliczeń	6
2.4. Zastosowania wzoru prostokątów	8
3. Projektowanie	9
3.1. Język programowania wykorzystany w projekcie	9
3.2. Narzędzia wykorzystane w projekcie	9
3.3. Biblioteki w projekcie	11
4. Implementacja	12
4.1. Struktura kodu	12
4.2. Rozłożenie metody calculate_partial()	13
4.3. Rozłożenie metody oblicz()	14
4.4. Efekt programu	16
4.5. Tabela z Excela wraz z wykresem	17
4.6. Network graph	18
4.7. Korzystanie z narzędzia Copilot	19
4.8. Problemy podczas pracy z Github Copilot	20
4.9. Pytania związane z Github Copilot	20
5. Wnioski	23
Literatura	24
Spis rysunków	25
Spis listingów	26

1. Ogólne określenie wymagań

Celem projektu jest stworzenie wielowątkowego programu w języku C++, który wylicza przybliżoną wartość liczby π metodą numerycznego całkowania oznaczonego funkcji. Projekt łączy teorię matematyczną z praktycznym wykorzystaniem nowoczesnych narzędzi programistycznych, takich jak wielowątkowość i zrównoleglenie obliczeń. Jest to zadanie, które nie tylko realizuje funkcje praktyczne, ale również pozwala na zbadanie wydajności algorytmu w zależności od liczby wątków oraz parametrów środowiska uruchomieniowego.

Wymagania programu:

- Program musi umożliwiać równoległe wykonywanie obliczeń matematycznych, przy czym użytkownik może skonfigurować liczbę wątków
- Program powinien obliczać wartość całki oznaczonej dla funkcji na przedziale od 0 do 1, przybliżając wartość liczby π
- Użytkownik musi mieć możliwość ustawienia liczby przedziałów w całkowaniu, co wpływa na dokładność wyniku.
- Zrównoleglenie wykonane za pomocą biblioteki "Thread".
- Program powinien wypisywać na ekran terminala czas liczenia całki wraz z wynikiem.
- Program powinien być napisany w języku C++.

Wymagania projektowe:

- Przeprowadzenie testów programu z różnymi liczbami z przedziału całki oznaczonej.
- Uruchomienie programu z ilością wątków od 1-50, a następnie zapisanie czasu pomiarów i liczby wątków w formie tabelarycznej w Excelu.
- Narysowanie wykresu przy pomocy arkusza kalkulacyjnego.
- Wygenerowanie dokumentacji Doxygen.

2. Analiza problemu

2.1. Ogólny problem

Celem programu jest obliczenie przybliżonej wartości liczby π poprzez rozwiązanie całki oznaczonej funkcji na przedziale $[0,1]$. Aby zrozumieć cały proces, konieczne jest omówienie matematycznej podstawy i technik numerycznych użytych w algorytmie.

Całka oznaczona

Liczba π może zostać wyznaczona z całki przy pomocy wzoru¹:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx.$$

Całka oznaczona pozwala na wyznaczenie pola pod krzywą funkcji $f(x) = \frac{4}{1+x^2}$ na przedziale $[0,1]$. Z matematycznego punktu widzenia jest to precyzyjne wyrażenie, ale w algorytmie przybliżamy wartość tej całki za pomocą metody prostokątów.

2.2. Metoda prostokątów

Metoda prostokątów jest prostą techniką numeryczną pozwalającą na obliczenie wartości przybliżonej całki przez podział przedziału $[0,1]$ na N równych części (przedziały całkowania). Zakładamy, że pole pod wykresem w każdym podprzedziale może być przybliżone przez pole prostokąta. Całkowite pole, czyli wynik całki, jest sumą pól wszystkich prostokątów.

¹Więcej na stronie Wikipedii[\[1\]](#)

Podział przedziału

Dzielimy przedział $[0,1]$ na N równych podprzedziałów, tak jak na następującym wzorze²:

$$\Delta x = \frac{1}{N}$$

Środek przedziału

Dla każdego prostokąta przyjmujemy wysokość funkcji $f(x)$, obliczaną w środku podprzedziału. Środek i -tego podprzedziału oznaczany jako x , jest obliczany ze wzoru poniżej³. Ten wzór pozwala na wyznaczenie współrzędnej środka każdego prostokąta.

$$x_i = (i + 0.5) \cdot \Delta x, \quad i = 0, 1, \dots, N - 1.$$

Wysokość prostokąta

Dla każdego x wyliczana jest wartość funkcji $f(x)$. Wysokość prostokąta odpowiada wartości funkcji w x . Pole prostokąta jest natępnie iloczynem jego wysokości oraz szerokości Δx . Funkcja została przedstawiona na poniższym wzorze⁴

$$f(x_i) = \frac{4}{1 + x_i^2}.$$

²Więcej na stronie Wikipedii[\[1\]](#)

³Więcej na stronie Wikipedii[\[1\]](#)

⁴Więcej na stronie Wikipedii[\[1\]](#)

Całkowita suma pól prostokątów

Sumując pola wszystkich prostokątów, otrzymujemy przybliżenie całki. Wzór przedstawia przybliżenie całki jako sumy, co umożliwia implementację w programie jako iterację po podprzedziałach. Przybliżona wartość liczby π jest obliczana jako⁵:

$$\pi \approx S = \Delta x \cdot \sum_{i=0}^{N-1} f(x_i).$$

2.3. Zrównoleglenie obliczeń

Przy dużej liczbie przedziałów (np 3'000'000'000), pojedyncze obliczenia mogą być czasochłonne. Aby przyspieszyć działanie algorytmu, należy podzielić zadanie na równoległe obliczenia przy użyciu wątków.

Podział pracy na wątki

Zakres obliczeń, czyli przedział $[0, N]$ jest dzielony między T wątków, gdzie T to liczba wątków. Każdy zakres pracy wątku t zaczyna się od indeksu⁶:

$$\text{start}_t = t \cdot \frac{N}{T}$$

Oraz kończy się na indeksie⁷:

$$\text{end}_t = (t + 1) \cdot \frac{N}{T}.$$

⁵Więcej na stronie Wikipedii[\[1\]](#)

⁶Więcej na stronie Wikipedii[\[1\]](#)

⁷Więcej na stronie Wikipedii[\[1\]](#)

Obliczenia w wątkach

Każdy wątek wykonuje sumowanie pól prostokątów w swoim zakresie, zgodnie ze wzorem który został przedstawiony na rysunku nr. Zostało to przedstawione na następującym wzorze⁸

$$S_t = \Delta x \cdot \sum_{i=\text{start}_t}^{\text{end}_t-1} f(x_i).$$

gdzie S_t to suma częściowa obliczona przez t -ty wątek.

Agregacja wyników

Po zakończeniu pracy wszystkich wątków, wyniki są sumowane w głównym wątku programu⁹:

$$S = \sum_{t=0}^{T-1} S_t.$$

Ostateczna wartość liczby π wynosi¹⁰:

$$\pi \approx S.$$

Przedstawiony algorytm wykorzystuje metodę prostokątów, aby przybliżyć wartość liczby π . Dzięki równolegleniu obliczeń możliwe jest wykorzystanie wielu rdzeni procesora, co pozwala skrócić czas działania. Proces podziału pracy między wątki oraz ich synchronizacja odgrywają kluczową rolę w wydajności programu. Numeryczne metody przybliżenia oraz techniki zrównoleglenia zostały przedstawione szczegółowo za pomocą wzorów i opisu kroków.

⁸Więcej na stronie Wikipedii[\[1\]](#)

⁹Więcej na stronie Wikipedii[\[1\]](#)

¹⁰Więcej na stronie Wikipedii[\[1\]](#)

2.4. Zastosowania wzoru prostokątów

Zastosowanie wzoru w praktyce

Metoda prostokątów znajduje szerokie zastosowanie w różnych dziedzinach nauki i inżynierii, zwłaszcza w sytuacjach, gdzie obliczenie wartości całki jest zbyt złożone lub niemożliwe w sposób analityczny. Wzór na wyznaczenie liczby π za pomocą całki oznaczonej, jak przedstawiono w poprzednim rozdziale, pozwala na obliczenie tej wartości w sposób przybliżony.

Zastosowanie wzoru w matematyce

Wzór jest przykładaniem zastosowania analizy numerycznej, w której przybliżone rozwiązania są stosowane do funkcji trudnych do zintegrowania bezpośrednio. Liczba π jest fundamentalną stałą matematyczną, a jego dokładne obliczenie odgrywa ważną rolę w teorii liczb, geometrii oraz analizie matematycznej. Dzięki tej metodzie możemy uzyskać wartości liczby π w sposób przybliżony, co jest kluczowe, gdy pracujemy w kontekście komputerowych obliczeń numerycznych, w których precyzja jest ograniczona.

Rozwój metod numerycznych

Wraz z rozwojem technologii obliczeniowych, metoda prostokątów, choć bardzo prosta, jest wciąż podstawą dla bardziej zaawansowanych algorytmów numerycznych. Na przykład:

- **Metoda trapezów** - wykorzystuje podobny podział przedziału, ale wysokość prostokąta jest zastąpiona średnią wartością funkcji na końcach przedziału. Zwiększa to dokładność obliczeń przy tej samej liczbie przedziałów.
- **Reguła Simpsona** - jest jeszcze dokładniejsza, wykorzystując interpolację kwadratową funkcji w każdym podprzedziale. To rozwiązanie jest stosowane, gdy potrzeba wysokiej precyzji przy mniejszych liczbach podprzedziałów.
- **Algorytmy Monte Carlo** - stosowane w przypadkach bardziej złożonych funkcji, w których klasyczne metody numeryczne stają się nieefektywne. Metoda prostokątów może być użyta jako wstępny etap w tego typu algorytmach, gdzie celem jest generowanie przybliżeń do wartości całki przez losowanie punktów.

3. Projektowanie

3.1. Język programowania wykorzystany w projekcie

W projekcie, zgodnie z wymaganiami, zostanie wykorzystany język C++. C++ jest doskonałym wyborem do implementacji złożonych algorytmów obliczeniowych, takich jak obliczanie przybliżonej wartości liczby PI metodą całkowania numerycznego. Język ten pozwala na ręczne zarządzanie pamięcią, co jest istotne przy pracy z dużymi zbiorami danych, jak w przypadku podziału przedziału całkowania na miliony elementów. Ponadto C++ oferuje możliwość programowania obiektowego, co umożliwia stworzenie klas do reprezentacji węzłów listy oraz samego algorytmu obliczeniowego. Dzięki tej strukturze kod będzie bardziej czytelny, łatwiejszy do utrzymania i rozwijania w przyszłości. C++ wspiera również wielowątkowość, co stanowi kluczowy aspekt projektu. Dzięki możliwości równoległego wykonywania obliczeń przy użyciu bibliotek standardowych, takich jak `pthread`, C++ pozwala na efektywne przetwarzanie dużych ilości danych. Taki wybór języka pozwoli również na precyzyjne zarządzanie zasobami systemowymi, co będzie miało duży wpływ na optymalizację czasu obliczeń.

3.2. Narzędzia wykorzystane w projekcie

Visual Studio 2022

W ramach realizacji projektu zastosowano Visual Studio 2022 jako główne środowisko programistyczne (IDE). Jest to jedno z najpopularniejszych narzędzi do pracy z językiem C++, oferujące szeroką gamę funkcji, które wspomagają programowanie, w tym:

- Zaawansowane narzędzia debugowania, które pozwalają na szybkie wykrywanie błędów w kodzie oraz monitorowanie zmiennych w trakcie jego wykonywania.
- Integrację z Git, co ułatwia zarządzanie wersjami kodu oraz kontrolowanie historii zmian.

Visual Studio 2022 pozwala na szybkie pisanie, kompilowanie i testowanie kodu, a także wspiera zaawansowane narzędzia do debugowania, które będą szczególnie przydatne w trakcie implementacji algorytmu obliczania PI. Dzięki tym funkcjom, praca nad projektem staje się bardziej efektywna, a kod łatwiejszy do zarządzania.

System kontroli wersji GIT

Do zarządzania kodem i śledzenia historii zmian w projekcie wykorzystywany jest system kontroli wersji Git. Git pozwala na bezpieczne przechowywanie kodu źródłowego i zarządzanie jego wersjami, co zapewnia pełną kontrolę nad projektem i umożliwia łatwe przywracanie wcześniejszych wersji kodu w przypadku wystąpienia problemów. Git jest rozproszonym systemem, co oznacza, że każda kopia repozytorium jest pełną wersją całego projektu, co zwiększa bezpieczeństwo danych. W projekcie będzie używana platforma GitHub do przechowywania kodu źródłowego w chmurze oraz zarządzania jego wersjami. GitHub oferuje dodatkowe funkcje, takie jak możliwość współpracy z innymi programistami, tworzenie pull requestów, zarządzanie gałęziami projektu (branchami) oraz śledzenie zgłoszeń (issues), co ułatwia zarządzanie projektem i jego dalszy rozwój.

Github Copilot

W projekcie planowane jest także wykorzystanie narzędzia GitHub Copilot, które jest wspierane przez sztuczną inteligencję. Copilot ułatwia proces programowania, sugerując fragmenty kodu na podstawie opisów funkcji, komentarzy w kodzie oraz kontekstu bieżącej pracy. Dzięki temu narzędziu możliwe będzie przyspieszenie procesu tworzenia kodu, zwłaszcza w przypadkach wymagających szybkiego generowania powtarzalnych lub rutynowych fragmentów kodu, takich jak implementacja algorytmów matematycznych. GitHub Copilot jest szczególnie przydatny w zadaniach, które wymagają codziennego powtarzania podobnych schematów kodu, jak np. operacje na tablicach, przetwarzanie danych wejściowych czy generowanie prostych struktur sterujących. Narzędzie to wykorzystuje technologię modelu językowego, który uczy się na podstawie ogromnych zbiorów danych, co pozwala na sugerowanie odpowiednich fragmentów kodu w zależności od kontekstu. Dzięki Copilotowi programista może zaoszczędzić czas, zwłaszcza przy pisaniu funkcji matematycznych czy algorytmów, co jest szczególnie przydatne w projektach obliczeniowych takich jak ten. Na przykład, podczas implementacji algorytmu obliczania liczby π metodą prostokątów, GitHub Copilot może automatycznie sugerować kod do obliczania wartości funkcji matematycznych, operacji na zmiennych czy pętli iterujących po dużych zbiorach danych. To narzędzie pozwala na szybsze osiągnięcie wyników przy zachowaniu wysokiej jakości kodu.

Arkusz kalkulacyjny Excel

Excel w projekcie będzie pełnił funkcję narzędzia do przechowywania i analizy danych uzyskanych podczas testowania algorytmu. Po napisaniu programu, w którym będzie ustalana liczba podziałów przedziału całki oznaczonej (np. 100'000'000, 1'000'000'000, 3'000'000'000) oraz liczba wątków (od 1 do 50), pomiary czasów wykonania algorytmu w różnych konfiguracjach zostaną zapisane w arkuszu kalkulacyjnym Excel. Excel pozwoli na organizowanie wyników w formie tabelarycznej, a następnie umożliwi wygenerowanie wykresu, który będzie przedstawiał zależność czasu wykonania programu od liczby używanych wątków. Oś Y wykresu będzie przedstawiać czas wykonania, natomiast oś X liczbę wątków. Po wykonaniu tych operacji, wykres będzie stanowił część dokumentacji projektu, ilustrującą wpływ równoległości na czas obliczeń.

3.3. Biblioteki w projekcie

`<cmath>`

Biblioteka ta w języku C++ dostarcza funkcji matematycznych, które są niezbędne do obliczeń związanych z całkowaniem numerycznym. W projekcie będzie wykorzystywana do obliczania funkcji matematycznych, które występują w algorytmie obliczania liczby π , zwłaszcza przy obliczaniu wartości funkcji $f(x) = \frac{4}{1+x^2}$, która pojawia się w metodzie prostokątów.

`<thread>`

Biblioteka ta w języku C++ umożliwia tworzenie i zarządzanie wątkami, co pozwala na równoległe wykonywanie obliczeń. Jest to niezbędne w projekcie, ponieważ algorytm obliczania liczby π będzie wykorzystywał równoległe obliczenia, co znacznie przyspieszy czas jego wykonania, zwłaszcza przy dużych liczbach podziału przedziału całkowania. Dzięki tej bibliotece możliwe jest podzielenie pracy na wiele wątków, co pozwala na jednoczesne obliczanie różnych części całki, a następnie zsumowanie wyników.

4. Implementacja

4.1. Struktura kodu

Projekt polega na obliczeniu przybliżonej wartości liczby π przy użyciu metody całkowania numerycznego, a dokładniej przy użyciu metody prostokątów, z równoległym przetwarzaniem obliczeń za pomocą wielu wątków. Został on zaprojektowany w taki sposób, aby umożliwić użytkownikowi wybór liczby przedziałów, na które podzielony zostanie przedział całkowania, oraz liczby wątków, które będą równolegle wykonywać obliczenia. Program jest podzielony na trzy pliki, co zapewnia modularność i ułatwia zarządzanie kodem:

Plik main.cpp

Plik main.cpp jest odpowiedzialny za interakcję z użytkownikiem i inicjowanie procesu obliczeniowego. Program zaczyna się od pobrania od użytkownika liczby przedziałów i liczby wątków, a następnie tworzy obiekt klasy Integral, wywołując metodę oblicz() do przeprowadzenia obliczeń. Po zakończeniu obliczeń, wynik oraz czas trwania obliczeń są wyświetlane.

Plik integral.cpp

Plik integral.cpp zawiera implementację klasy Integral, która odpowiada za przeprowadzenie obliczeń. Klasa ta posiada konstruktor, który przyjmuje liczbę przedziałów oraz liczbę wątków. Dodatkowo, zawiera metody do obliczania części całki przez poszczególne wątki oraz metody pomocnicze do sumowania wyników.

Plik integral.h

Plik integral.h to plik nagłówkowy, w którym zdefiniowana jest klasa Integral i jej interfejs. Zawiera deklaracje metod oraz prywatnych zmiennych, takich jak liczba przedziałów, liczba wątków, wynik π oraz czas trwania obliczeń.

4.2. Rozłożenie metody `calculate_partial()`

Metoda `calculate_partial()` jest odpowiedzialna za obliczanie części całki oznaczonej dla zadanej liczby przedziałów w przedziale $[0,1]$. Metoda ta realizuje numeryczne całkowanie oznaczone za pomocą metody prostokątów dla fragmentu przedziału całkowania. Pracuje tylko na części danych, zdefiniowanych przez parametry wejściowe `start` i `end`. Wynik tej częściowej sumy zostanie później połączony z wynikami innych wątków, aby uzyskać wartość całki dla całego przedziału. Metoda ta została przedstawiona na Listingu nr. 1 (s. 13).

```
1 double Integral::calculate_partial(long long int start, long long
   int end)
2 {
3     double dx = 1.0 / liczba_przedzialow;
4     double sum = 0;
5     for (long long int i = start; i < end; i++)
6     {
7         double x = (i + 0.5) * dx;
8         sum += 4.0 / (1.0 + x * x);
9     }
10    return sum;
11 }
```

Listing 1. Metoda `calculate_partial()`

Szczegółowy opis metody przedstawionej na Listingu nr. 1 (s. 13).

- Obliczanie długości pojedynczego podprzedziału na podstawie zmiennej. Jest to krok dx który jest równy szerokości prostokątów używanych w metodzie prostokątów. (Linia 3)
- Inicjalizacja zmiennej `sum` która będzie przechowywać sumę wartości funkcji w przedziałach. (Linia 4)
- Pętla `for` przechodzi przez wszystkie przedziały w zakresie `[start, end]`. Dla każdego przedziału obliczane jest pole odpowiadającego mu prostokąta, które następnie dodawane jest do sumy. (Linia 5-9)
- Dla każdego podprzedziału obliczany jest środek podprzedziału. (Linia 7)
- Obliczana jest wartość funkcji $F(x)$. (Linia 8)
- Zwracana jest wartość obliczonej sumy. (Linia 10)

4.3. Rozłożenie metody oblicz()

Metoda `oblicz()`, jest odpowiedzialna za równoległe obliczanie wartości całki oznaczonej w celu przybliżenia liczby π . Realizuje to, dzieląc obliczenia na wiele wątków, z których każdy przetwarza część całkowitego zakresu. Została ona przedstawiona na Listingu nr. 2 (s. 14).

```
1 void Integral::oblicz()
2 {
3     auto start = std::chrono::high_resolution_clock::now();
4     std::vector<std::thread> threads;
5     std::vector<double> results(liczba_watkow);
6     for (int i = 0; i < liczba_watkow; i++)
7     {
8         long long int start = i * liczba_przedzialow / liczba_watkow;
9         long long int end = (i + 1) * liczba_przedzialow /
10            liczba_watkow;
11
12         threads.push_back(std::thread([this, start, end, i, &results]()
13             {
14                 results[i] = calculate_partial(start, end);
15             }));
16     }
17     for (auto& t : threads)
18     {
19         t.join();
20     }
21
22     wynik_pi = 0;
23     for (int i = 0; i < liczba_watkow; i++)
24     {
25         wynik_pi += results[i];
26     }
27     wynik_pi *= 1.0 / liczba_przedzialow;
28     auto end = std::chrono::high_resolution_clock::now();
29     czas_obliczen = std::chrono::duration<double>(end - start).count
30         ();
31 }
```

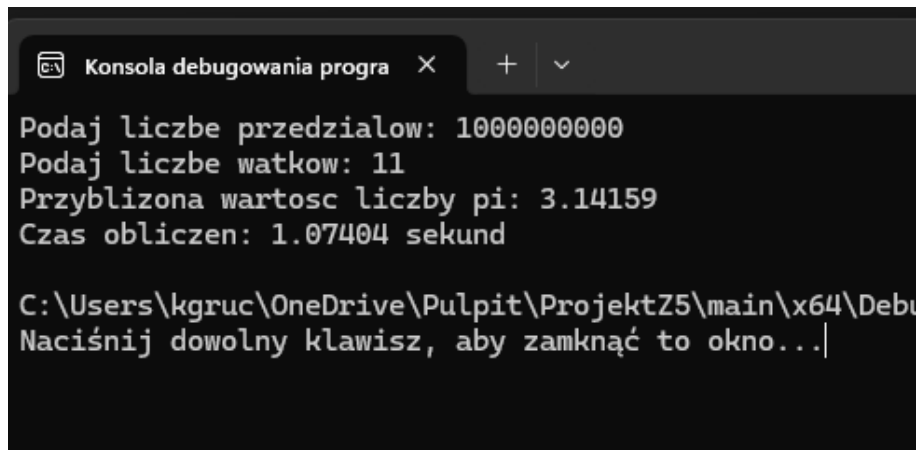
Listing 2. Metoda `calculate_partial()`

Szczegółowy opis metody przedstawionej na Listingu nr. 2 (s. 14).

- Pomiar czasu rozpoczęcia obliczeń, zapisuje aktualny czas systemowy na początku działania metody. Używany do późniejszego wyliczenia czasu trwania obliczeń. (Linia 3)
- Inicjalizacja wektorów do zarządzania wątkami i wynikami. `Threads` to wektor który przechowuje obiekty wątków, a `results` to wektor który przechowuje wynik częściowy całki wyliczonej przez każdy wątek. (Linia 4,5)
- Tworzenie wątków i przydzielanie zakresów obliczeń. Pętla `for` tworzy wątki, dla każdego wątku wylicza przydzielony zakres przedziałów. `std::thread` tworzy nowy wątek który wykounje funkcje anonimową. Funkcja ta wywołuje metodę `calculate_partial` z przydzielonym zakresem i zapisuje wynik do wektora `results`. (Linia 6-15)
- Oczekiwanie na zakończenie pracy wszystkich wątków, Główny wątek programu czeka, aż każdy z utworzonych wątków zakończy działanie. Bez tego metoda mogłaby przejść do następnych kroków, zanim wątki zakończą obliczenia. (Linia 16-19)
- Sumowanie wyników częściowych. Iteruje przez wektor `results` i sumuje wyniki częściowe obliczone przez wątki. Ostateczna wartość `wynik_pi` zawiera sumę całki wyliczoną metodą prostokątów. (Linia 22-25)
- Wynik zostaje skalowany. Dostosowanie wyniku do rzeczywistej wartości całki. (Linia 26)
- Pomiar czasu zakończenia obliczeń. Zapisuje czas zakończenia obliczeń i oblicza różnicę czasu między początkiem a końcem. (Linia 27,28)

4.4. Efekt programu

Efektom napisanego kodu jest program w którym użytkownik podaje ilość przydziałów, a następnie ilość wątków. Program podaje obliczoną liczbę pi oraz czas ile zajęło wyliczenie. Na rysunku nr. 4.1 (s. 16) został przedstawiony działający program.

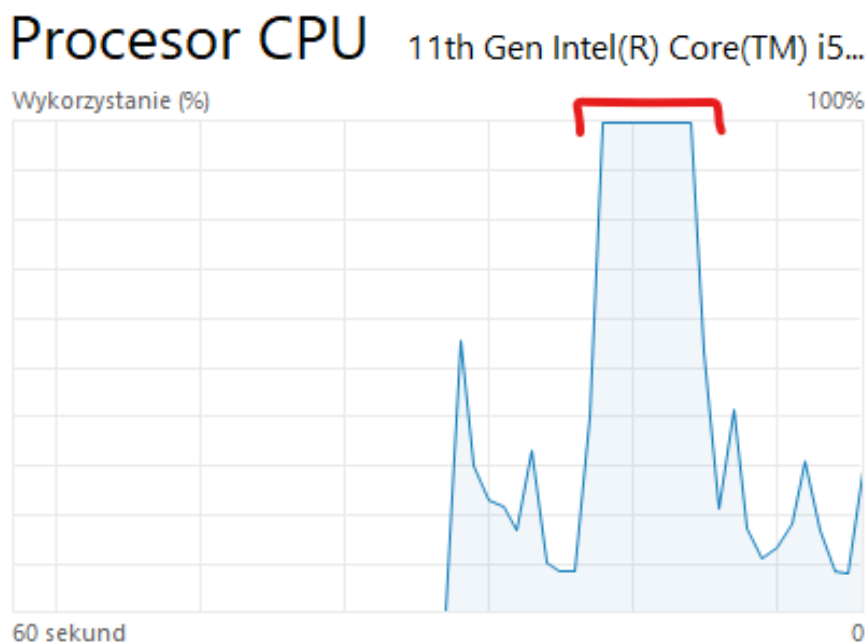


```
Konsola debugowania progra x + v
Podaj liczbe przedzialow: 1000000000
Podaj liczbe watkow: 11
Przyblizona wartosc liczby pi: 3.14159
Czas obliczen: 1.07404 sekund

C:\Users\kgruc\OneDrive\Pulpit\ProjektZ5\main\x64\Debu
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Rys. 4.1. Efekt programu

Program ten jest "zasobożerny", na rysunku nr. 4.2 (s. 16) można zobaczyć wykres z menadżera zadań który przedstawia wykorzystanie procesora w danym momencie, kolorem czerwonym został zaznaczony moment w którym program wykonywał obliczenia dla 5'000'000'000 przedziałów i 9 wątków.



Rys. 4.2. Menadżer zadań - zużycie procesora

4.5. Tabela z Excela wraz z wykresem

Tabela

Na rysunku nr. 4.3 (s. 17) oraz nr. 4.4 (s. 17) możemy zobaczyć tabelę która przedstawia testy dla trzech różnych przedziałów: 100'000'000, 1'000'000'000, 3'000'000'000. Każdy przedział był testowany na wątkach z zakresu od 1 - 50, w tabelce widzimy czasy ile zajęło obliczenie danego przedziału dla danej liczby wątków.

Wątki	10000000	1000000000	3000000000
1	1.010	9.898	30.100
2	0.504	5.012	15.840
3	0.344	3.423	11.123
4	0.262	2.625	8.016
5	0.213	2.114	6.475
6	0.177	1.778	5.488
7	0.154	1.539	4.884
8	0.136	1.353	4.226
9	0.121	1.233	3.696
10	0.110	1.092	3.369
11	0.104	1.003	3.071
12	0.108	0.961	3.179
13	0.120	1.074	3.196
14	0.127	1.067	3.184
15	0.144	1.036	3.120
16	0.137	1.046	3.166
17	0.129	1.037	2.959
18	0.122	1.091	2.940
19	0.117	1.005	3.133
20	0.112	1.048	3.335
21	0.111	1.061	3.214
22	0.103	1.092	3.146
23	0.124	1.046	3.378
24	0.098	1.135	3.025
25	0.125	1.113	3.292

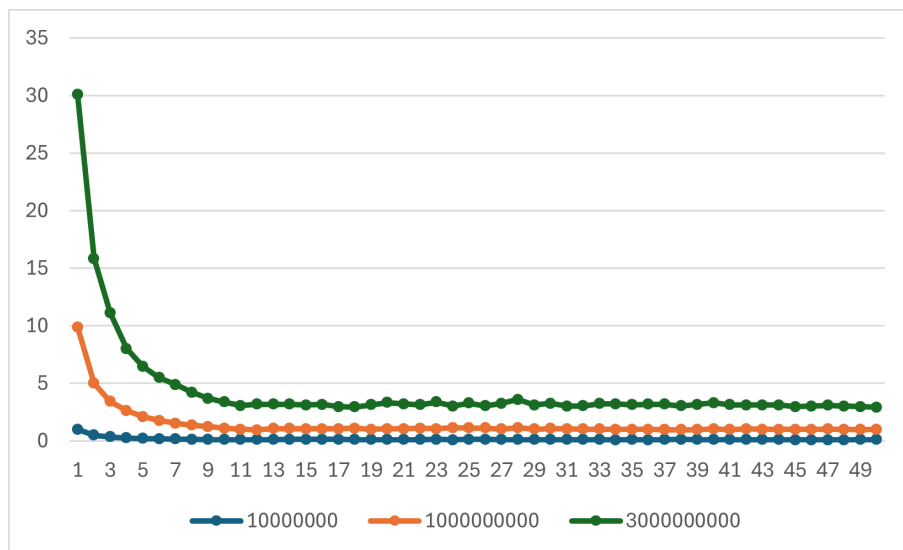
Rys. 4.3. Tabela część 1

Wątki	10000000	1000000000	3000000000
26	0.124	1.127	3.044
27	0.121	1.029	3.253
28	0.120	1.142	3.585
29	0.113	1.016	3.115
30	0.111	1.069	3.263
31	0.108	1.043	3.017
32	0.104	1.025	3.037
33	0.103	1.035	3.235
34	0.100	1.010	3.212
35	0.102	0.998	3.133
36	0.099	0.991	3.190
37	0.119	0.980	3.186
38	0.133	0.979	3.052
39	0.113	0.987	3.154
40	0.110	1.043	3.299
41	0.108	0.980	3.140
42	0.106	1.041	3.111
43	0.103	1.011	3.119
44	0.104	1.010	3.101
45	0.101	1.004	2.981
46	0.098	1.002	3.028
47	0.099	1.026	3.080
48	0.097	1.009	3.007
49	0.112	0.985	2.976
50	0.110	1.003	2.918

Rys. 4.4. Tabela część 2

Wykres

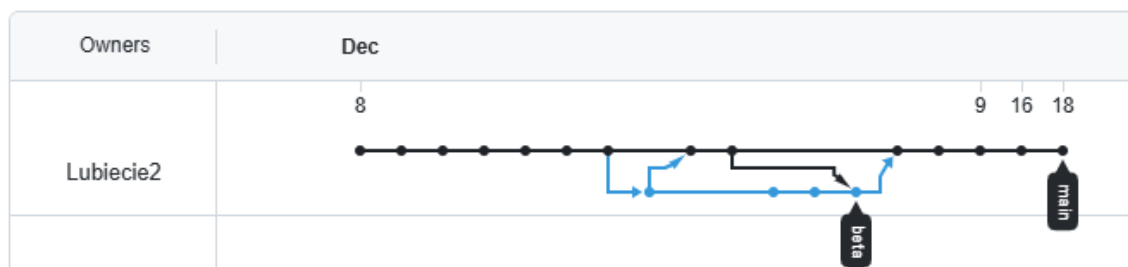
Rysunek nr. 4.5 (s. 18) przedstawia wykres który pochodzi z programu excel, wykres ten powstał na podstawie tabeli która jest na stronie nr. 17, widzimy na niej czas który się zmniejsza wraz ze zwiększeniem ilości wątków.



Rys. 4.5. Wykres

4.6. Network graph

Na rysunku nr. 4.6 (s. 18) zostało przedstawione drzewo commitów z platformy Github (network graph), na którym widać commity, operacje scalania oraz daty kiedy były wysyłane commity na zdalne repozytorium.



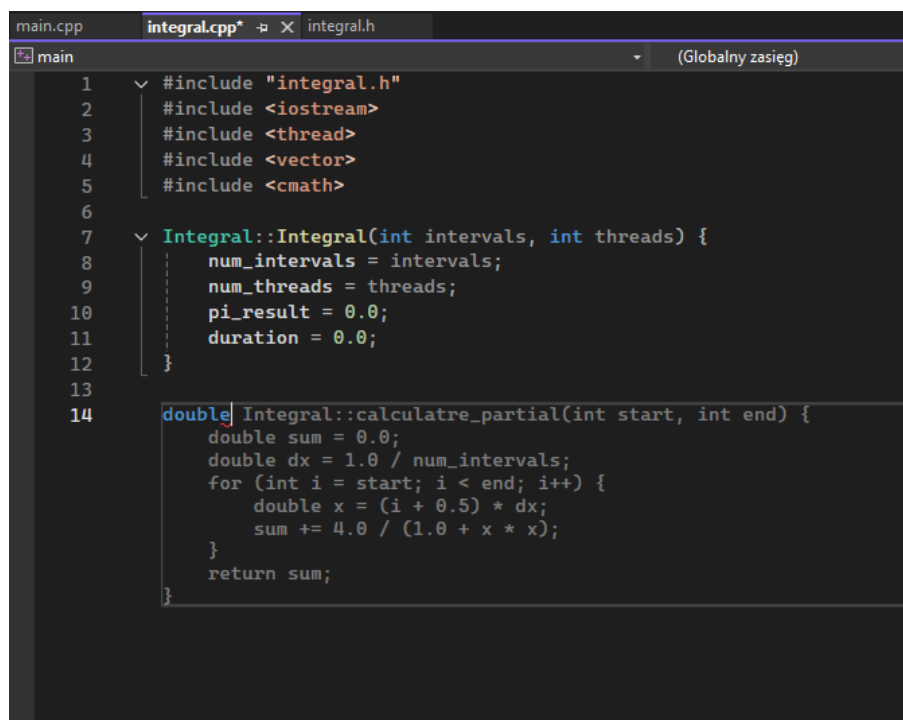
Rys. 4.6. Drzewo Network Graph

4.7. Korzystanie z narzędzia Copilot

Narzędzie Copilot był istotnym wsparciem podczas realizacji tego projektu w kilku kluczowych obszarach:

- **Generowanie szkieletu kodu** - Narzędzie ułatwiało tworzenie wstępnych wersji funkcji i klas na podstawie opisów w komentarzach. Na przykład, Copilot pomógł w szybkim wygenerowaniu szkieletu klasy `Integral`, w tym deklaracji metod i atrybutów, co pozwoliło skupić się na implementacji logiki algorytmu.
- **Automatyzacja powtarzalnych fragmentów kodu** - W projekcie konieczne było wielokrotne implementowanie podobnych struktur, takich jak pętle i operacje matematyczne. Copilot sugerował gotowe fragmenty kodu, dzięki czemu proces programowania przebiegał szybciej i był mniej podatny na błędy.
- **Tworzenie komentarzy i dokumentacji** - Narzędzie wspierało również generowanie dokumentacji w stylu Doxygen. Na podstawie nazwy metod i atrybutów proponowało odpowiednie opisy.

Na rysunku nr. 4.7 (s. 19) został przedstawiony fragment programu Visual Studio 2022 w którym została zainstalowana wtyczka Github Copilot, na rysunku można zobaczyć jak Copilot pomaga w pisaniu kodu poprzez sugerowanie linii kodu.



```
main.cpp | integral.cpp* | integral.h
main
1  #include "integral.h"
2  #include <iostream>
3  #include <thread>
4  #include <vector>
5  #include <cmath>
6
7  Integral::Integral(int intervals, int threads) {
8      num_intervals = intervals;
9      num_threads = threads;
10     pi_result = 0.0;
11     duration = 0.0;
12 }
13
14 double Integral::calculatre_partial(int start, int end) {
    double sum = 0.0;
    double dx = 1.0 / num_intervals;
    for (int i = start; i < end; i++) {
        double x = (i + 0.5) * dx;
        sum += 4.0 / (1.0 + x * x);
    }
    return sum;
}
```

Rys. 4.7. Github Copilot w praktyce

4.8. Problemy podczas pracy z Github Copilot

Podczas pracy z narzędziem Github Copilot występowały różne problemy takie jak np wygenerowanie błędów w kodzie, użył złej zmiennej co doprowadziło do tego że nie można było wprowadzić więcej jak 2mld przedziałów i trzeba było to ręcznie poprawiać. Kolejnym błędem jaki popełnił to niedokładność w obliczeniach matematycznych, Copilot sugerował użycie zmiennej typu float zamiast double co prowadziło do utraty precyzji przy dużej liczbie iteracji

4.9. Pytania związane z Github Copilot

Jak generowany jest kod?

Kod w tym programie był generowany narzędziem Github Copilot na podstawie poleceń i opisów dostarczonych przez użytkownika. Większość fragmentów kodu które wygenerowało AI było poprawnych merytorycznie ale wymagały poprawek przez użytkownika, niektóre fragmenty kodu wymagały doprecyzowania pytań.

Czy kod się kompiluje?

Kod kompiluje się poprawnie po wprowadzeniu drobnych poprawek użytkownika, np. Copilot zrobił taki błąd jak użycie złego typu zmiennych (int zamiast long long int) przez co nie było możliwe wykonywanie obliczeń na przedziałach powyżej 2mld.

Czy kod się uruchamia?

Tak, kod uruchamia się poprawnie, nie było problemów w trakcie uruchamiania.

Czy nie ma błędów podczas kompilacji?

Podczas kompilacji nie było większych błędów które uniemożliwiały kompilacje, ale wystąpiły drobne błędy które można było szybko naprawić. Najczęściej chodziło o błąd związany z przekroczeniem zakresu typów danych.

Czy nie ma rażących błędów, np. wycieków pamięci?

Nie, w moim kodzie nie zauważyłem błędów związanych z np wyciekami danych, czy innymi podobnymi błędami. AI generuje kod na podstawie dostępnych wzorców, co oznacza, że może popełnić błędy, takie jak wycieki pamięci, błędne zarządzanie wskaźnikami lub brak zamykania otwartych zasobów, ale w moim programie nie zdarzyły się takie błędy.

Ile razy trzeba było generować kod?

Kod musiał zostać kilkukrotnie, głównie ze względu na dobre błędy które wynikały ze złego doprecyzowania zapytania przez użytkownika, oraz dla porównania czy w innym przypadku kod będzie wygenerowany taki sam, czy może będą wprowadzone jakieś poprawki które usprawnią program.

Czy dużo było poprawy kodu ?

Dużo to złe określenie, ale w trakcie pisania kodu trzeba było wprowadzić kilka poprawek, głównie związanych z doprecyzowaniem pytania, co wymagało ponownego generowania kodu.

Czy poprawnie dołączone są biblioteki?

Tak Copilot poprawnie dołączał biblioteki które były niezbędne w tym programie np biblioteka związana z wątkami czy biblioteka do zarządzania czasem. Biblioteki były zgodne z wymaganiami programu i nie wystąpiły problemy związane z ich działaniem.

Czy dostajemy poprawne wyniki?

Tak program który został wygenerowany przy pomocy Copilota zwracał poprawne wyniki. Po przeprowadzeniu testów, obliczenia całek oznaczonych oraz przybliżonej liczby π były zgodne z oczekiwaniami.

Zagrożenia przy korzystaniu z AI

AI może generować nieoptymalny lub błędny kod, co prowadzi do strat czasu na poprawki. Nadmierne poleganie na AI może ograniczyć rozwój umiejętności programisty. Niektóre fragmenty kodu mogą być przypadkowo generowane na podstawie wzorców objętych prawami autorskimi

Czy należy korzystać z AI przy pisaniu programów?

Tak, ale z rozważą. AI to potężne narzędzie wspomagające, które przyspiesza proces tworzenia kodu i pomaga w prostych, powtarzalnych zadaniach. Jednak nie zastępuje w pełni programisty, który musi weryfikować i optymalizować kod

W czym AI pomaga, a w czym przeszkadza?

AI zdecydowanie pomaga w: generowaniu szkieletów kodu, sugerowanie funkcji i bibliotek, przyspieszenie pracy poprzez podpowiedzi kontekstowe. AI przeszkadza kiedy: Błędnie sugeruje rozwiązania lub są nieefektywne, wygenerowany kod czasem wymaga dużej liczby poprawek, brak rozumienia intencji użytkownika w bardziej złożonych przypadkach.

Jak muszą być definiowane polecenia, aby AI poprawnie wygenerowała kod?

Polecenia muszą być: jasne i precyzyjne, określające, co kod ma zrobić i jakie są jego wejścia/wyjścia, zawierać szczegóły, takie jak wymagane biblioteki czy konwencje.

Czy może programować osoba nieznająca się na programowaniu (na chwilę obecną)?

AI może pomóc osobie początkującej w stworzeniu prostych aplikacji, ale osoba taka nadal musi posiadać podstawową wiedzę o strukturze programów, językach programowania i kompilacji. Bez tej wiedzy trudno będzie weryfikować poprawność kodu.

Jaka jest różnica między czatem a GitHub Copilot?

Copilot jest narzędziem który jest kierowany głównie do programistów, generuje on kod na podstawie kontekstu w edytorze i jest przeznaczony do pracy w czasie rzeczywistym nad projektem, a GPT to bardziej uniwersalne narzędzie które umożliwia rozmowę w języku naturalnym, wyjaśnia koncepcję i generuje fragmenty.

5. Wnioski

Projekt realizowany przy użyciu narzędzi takich jak GitHub Copilot, C++, Visual Studio oraz Excel pozwolił na efektywne przeprowadzenie obliczeń numerycznych przy zastosowaniu metody prostokątów. Proces ten pozwolił nie tylko na przybliżenie wartości liczby π , ale również na analizę wydajności wielowątkowego podejścia do rozwiązywania problemów obliczeniowych. Z przeprowadzonych testów oraz analiza wyników umożliwiły wyciągnięcie następujących wniosków:

- **Efektywność wielowątkowa** - Zastosowanie wielowątkowości znacząco wpłynęło na skrócenie czasu obliczeń. Wyniki wskazują, że zwiększenie liczby wątków poprawia wydajność algorytmu, jednakże przy pewnym poziomie liczba wątków przestaje proporcjonalnie przyspieszać obliczenia z powodu ograniczeń sprzętowych i narztu związanego z synchronizacją wątków
- **Znaczenie precyzji danych** - Korzystanie z odpowiednich typów danych, takich jak `double` oraz `long long int`, było kluczowe dla zachowania dokładności obliczeń przy dużej liczbie przedziałów. Błędy w deklaracjach zmiennych mogłyby skutkować ograniczeniem zakresu obliczeń, a nawet błędami w działaniu programu.
- **Wykorzystanie narzędzi wspierających programowanie** - GitHub Copilot okazał się przydatnym wsparciem w procesie programowania, szczególnie przy generowaniu powtarzalnych fragmentów kodu, takich jak implementacja algorytmów matematycznych czy wielowątkowości. Jednak narzędzie wymagało ciągłej weryfikacji wygenerowanego kodu, ponieważ generowało błędy, takie jak nieodpowiednie typy zmiennych czy brak synchronizacji wątków. To podkreśla, że korzystanie z AI nie zastępuje wiedzy programistycznej, a jedynie ją wspomaga.
- **Prezentacja wyników i analiza danych** - Wykorzystanie programu Excel do analizy wyników pomiarów i wizualizacji danych było kluczowe dla zrozumienia zależności między liczbą wątków a czasem obliczeń. Graficzne przedstawienie wyników w postaci wykresu pozwoliło na łatwe zauważenie punktu, w którym zwiększanie liczby wątków przestaje przynosić znaczące korzyści.

Bibliografia

- [1] *Wikipedia*. URL: <https://pl.wikipedia.org/wiki/Ca%C5%82ka> (term. wiz. 09.12.2024).
- [2] *Serwis edukacyjny*. URL: https://eduinf.waw.pl/inf/alg/004_int/0002.php (term. wiz. 10.12.2024).
- [3] *Algorytm Edu*. URL: <https://www.algorytm.edu.pl/algorytmy-maturalne/metoda-prostokatow.html> (term. wiz. 10.12.2024).
- [4] *Github Copilot*. URL: <https://github.com/features/copilot> (term. wiz. 11.12.2024).
- [5] *Microsoft Excel*. URL: <https://www.microsoft.com/pl-pl/microsoft-365/excel> (term. wiz. 16.12.2024).

Spis rysunków

4.1. Efekt programu	16
4.2. Menadżer zadań - zużycie procesora	16
4.3. Tabelka część 1	17
4.4. Tabelka część 2	17
4.5. Wykres	18
4.6. Drzewo Network Graph	18
4.7. Github Copilot w praktyce	19

Spis listingów

1.	Metoda <code>calculate_partial()</code>	13
2.	Metoda <code>calculate_partial()</code>	14