

# AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich  
Katedra Informatyki

## DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

### **Obliczanie wartości liczby $\pi$ z zastosowaniem Chat GPT**

Autor:  
Michał Bernardy

Prowadzący:  
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

# Spis treści

<b>1. Ogólne określenie wymagań</b>	<b>4</b>
1.1. Wymagania projektu . . . . .	5
<b>2. Analiza problemu</b>	<b>6</b>
2.1. Opis algorytmu obliczania liczby $\pi$ metodą całkowania numerycznego . .	6
2.2. Szczegóły implementacji . . . . .	6
2.3. Działanie programu . . . . .	7
2.4. Testy algorytmu . . . . .	8
2.5. Opis algorytmu i programu poprzez ChatGPT . . . . .	8
2.5.1. Program . . . . .	9
2.5.2. Optymalizacja równoległa . . . . .	10
2.5.3. Podsumowanie . . . . .	10
<b>3. Projektowanie</b>	<b>11</b>
3.1. Wykorzystywane narzędzia . . . . .	11
3.1.1. C++ . . . . .	11
3.1.2. Visual Studio . . . . .	11
3.1.3. Biblioteka „thread” . . . . .	11
3.1.4. Excel . . . . .	12
3.1.5. Git . . . . .	12
3.1.6. GitHub . . . . .	13
3.1.7. ChatGPT . . . . .	13
<b>4. Implementacja</b>	<b>14</b>
4.1. Obliczanie wartości liczby Pi . . . . .	14
4.2. Obliczanie segmentu całki . . . . .	15
4.3. Funkcja robocza do obliczeń . . . . .	15
4.4. Czas wykonania obliczeń . . . . .	16
4.5. Analiza wyników i wpływ liczby wątków na czas wykonania . . . . .	16
4.6. Przebieg procesu tworzenia programu . . . . .	18
4.6.1. Generowanie kodu . . . . .	18

4.6.2.	Kompletność kodu i kompilacja . . . . .	18
4.6.3.	Wykrywanie błędów . . . . .	18
4.6.4.	Testowanie i poprawność wyników . . . . .	18
4.6.5.	Podsumowanie procesu . . . . .	19
<b>5.</b>	<b>Wnioski</b>	<b>20</b>
5.1.	Algorytm obliczania liczby Pi . . . . .	20
5.2.	Zastosowanie wątków i równoległości . . . . .	20
5.3.	Git i GitHub . . . . .	20
5.4.	Wnioski z wykorzystania AI przy pisaniu programu . . . . .	21
5.4.1.	Zagrożenia przy korzystaniu z AI . . . . .	21
5.4.2.	Zalety korzystania z AI . . . . .	21
5.4.3.	Formułowanie poleceń dla AI . . . . .	22
5.4.4.	Czy AI może zastąpić programistów? . . . . .	22
5.4.5.	Różnica między czatem AI a GitHub Copilot . . . . .	22
5.5.	Podsumowanie . . . . .	23
	<b>Literatura</b>	<b>24</b>
	<b>Spis rysunków</b>	<b>25</b>
	<b>Spis listingów</b>	<b>26</b>

## 1. Ogólne określenie wymagań

Napisz program, który wylicza przybliżoną liczbę PI metodą całkowania numerycznego całki oznaczonej z funkcji. Program winien mieć możliwość ustawienia przez użytkownika ilości liczb z przedziału całki oznaczonej oraz ustawienia ilości wątków celem zrównoleglenia obliczeń matematycznych. Zrównoleglenie wykonać za pomocą biblioteki thread, która działa w systemach z standardem POSIX. Program musi być napisany w języku C++. Program powinien wypisywać na ekran terminala czas liczenia całki wraz z wynikiem. Poproś czata o opisanie jak działa algorytm oraz program, który wygenerował.

Po napisaniu programu należy przeprowadzić testy algorytmu. W programie ustala się ilość liczb podziału z przedziału całki oznaczonej np. 100'000'000, 1'000'000'000, 3'000'000'000. Następnie uruchamia się program z ilością wątków od 1 do 50. Czasy z pomiarów i liczba wątków zapisać w formie tabelarycznej w Excelu.

Po wykonaniu pomiarów, wyniki zapisane do arkusza kalkulacyjnego pozwalają narysować wykres, gdzie oś Y będzie to czas, a oś X będą to wątki (powinniśmy dostać trzy linie na wykresie). Rysunek wykresu wklej do dokumentacji.

Podczas testowania w systemie Windows otwórz menedżer zadań i sprawdzaj zajętość procesora, zajętość rdzeni, ilość wątków, częstotliwość procesora, itp. Możesz kilka rysunków wkleić do dokumentacji.

Kolejnym krokiem jest sprawdzenie, czy zmniejsza się czas pracy programu, gdy zwiększamy ilość wątków. Jeśli tak, to mamy program dobrze napisany. Jeśli nie, to powinniśmy szukać błędu algorytmicznego. Zobacz, jaki najmniejszy czas dostaniesz przy jakiej ilości wątków. Czy parametry twojego komputera mają znaczenie? Jeśli masz jakiś inny komputer, to sprawdź, czy te same wyniki dostaniesz.

Celem projektu jest napisanie programu wielowątkowego z wykorzystaniem czata. Podczas pisania programu należy odpowiedzieć na pytania takie jak: jak jest generowany kod (poprawność merytoryczna), czy kod się kompiluje, czy kod się uruchamia, czy nie ma błędów podczas kompilacji, czy nie ma rażących błędów np. wycieków pamięci, ile razy trzeba było generować kod, czy dużo było poprawy kodu, czy poprawnie dołączone są biblioteki, czy dostajemy poprawne wyniki (jeśli nie, to spróbujmy, aby chat znalazł błąd i go poprawił). Jeśli dalej wyniki wychodzą błędnie, to spróbuj, aby chat przedstawił, jakie mogą być przyczyny źle działającego programu. Przy problemach z uruchomieniem programu skorzystaj z podpowiedzi, które będzie sugerowała AI.

Po napisaniu programu zastanów się i odpowiedz na pytania: jakie są zagrożenia

przy korzystaniu z AI, czy należy korzystać z AI przy pisaniu programów, w czym AI pomaga, a w czym przeszkadza. Jak muszą być definiowane polecenia, aby AI poprawnie wygenerowała kod, czy może programować osoba nieznająca się na programowaniu (na chwilę obecną), jaka jest różnica między czatem a GitHub Copilot? Poprzedni projekt był z zastosowaniem GitHub Copilot, więc wiesz jak się z niego korzysta oraz jakie ma wady i zalety. Można spróbować generować program z użyciem innych czatów lub wersji (tylko w dokumentacji napisz który to chat). Można w dokumentacji zaprezentować ciekawe dialogi prowadzone z AI. Zastanów się jak może wyglądać przyszłość AI.

### 1.1. Wymagania projektu

Do projektu należy napisać dokumentację w  $\text{\LaTeX}$ , DoxyGen oraz zapisać projekt w serwisie GitHub. Chciałbym, aby w dokumentacji pochylić się nad takimi przykładowymi pytaniami jak są:

- Jak generować poprawny kod w języku C++?
- Jak efektywnie wykorzystać wielowątkowość?
- Jakie są problemy z zrównoleglaniem obliczeń numerycznych?
- Jakie narzędzia i biblioteki mogą pomóc w realizacji takiego projektu?
- Jakie mogą być optymalizacje algorytmu?

## 2. Analiza problemu

### 2.1. Opis algorytmu obliczania liczby $\pi$ metodą całkowania numerycznego

Obliczanie liczby  $\pi$  jest klasycznym zagadnieniem w analizie numerycznej, które pozwala na przybliżenie tej liczby poprzez obliczenie całki oznaczonej. W tym przypadku, celem jest obliczenie wartości liczby  $\pi$  za pomocą metody całkowania numerycznego z wykorzystaniem wątków do zrównoleglenia obliczeń. Aby to osiągnąć, należy obliczyć całkę z funkcji  $\frac{4}{1+x^2}$  na przedziale  $[0, 1]$ , który jest związany z obliczaniem wartości liczby  $\pi$ . Metoda całkowania numerycznego przy użyciu tzw. metody prostokątów (lub metody trapezów) pozwala na przybliżenie wartości całki przez podzielenie przedziału na  $n$  równych części i sumowanie wartości funkcji w tych punktach. W celu zwiększenia dokładności obliczeń, algorytm podzieli przedział na dużą liczbę kawałków (np. 100'000'000, 1'000'000'000, 3'000'000'000), co daje większą precyzję obliczeń.

Zastosowanie wielu wątków umożliwia równoczesne obliczanie podprzedziałów, co pozwala na znaczące skrócenie czasu wykonania całego obliczenia. Liczba wątków będzie mogła być ustawiana przez użytkownika programu, co pozwala na testowanie różnych konfiguracji sprzętowych i sprawdzanie wpływu liczby rdzeni procesora na czas obliczeń. Użycie biblioteki „thread” w systemach z standardem POSIX pozwala na łatwe zrównoleglenie obliczeń w C++.<sup>1</sup>

### 2.2. Szczegóły implementacji

Program będzie wymagał od użytkownika podania dwóch parametrów wejściowych:

- **Liczba podziałów przedziału całkowania** – liczba dzieląca przedział  $[0, 1]$  na  $n$  równych części. Im większa liczba, tym większa dokładność obliczeń, ale także większy czas wykonania.
- **Liczba wątków** – liczba wątków, które będą równolegle obliczały poszczególne części całki.

Algorytm dzieli cały przedział całkowania na  $n$  części, gdzie każda część będzie obliczana przez oddzielny wątek. Wątek i jego zakres przedziału są przydzielane na podstawie całkowitej liczby wątków. Każdy wątek oblicza sumę funkcji w swojej

---

<sup>1</sup>wikipedia.com [1].

części przedziału, a następnie wyniki są sumowane, aby uzyskać całkowitą wartość przybliżoną liczby  $\pi$ .

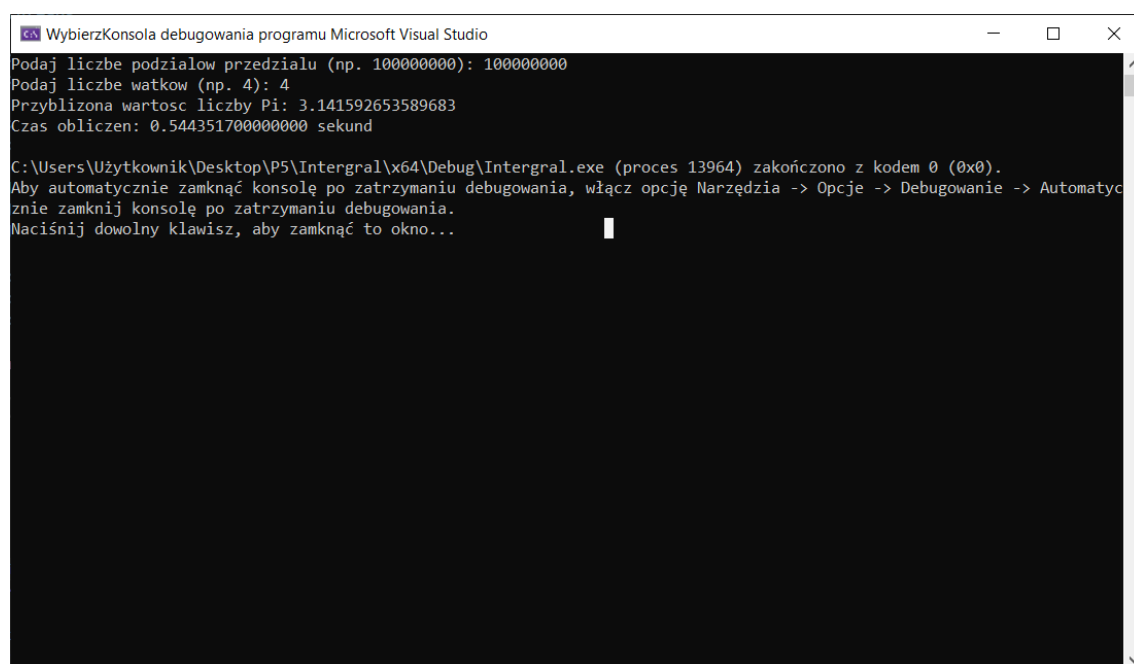
Po zakończeniu obliczeń przez wszystkie wątki, czas wykonania całkowity jest wyświetlany na ekranie terminala, a użytkownik otrzymuje wynik przybliżenia liczby  $\pi$ .

## 2.3. Działanie programu

Po uruchomieniu programu użytkownik wprowadza:

- Liczbę podziałów całki, np. 100'000'000,
- Liczbę wątków, np. od 1 do 50.

Program będzie uruchamiany kilkakrotnie z różnymi konfiguracjami liczby wątków (od 1 do 50), a wyniki zostaną zapisane w formie tabeli. Pomiar czasu będzie dokonywany za pomocą funkcji zegara systemowego, a także mierzony będzie czas wykonania programu przy różnych liczbach podziałów całki 7 2.1.



```
WybierzKonsola debugowania programu Microsoft Visual Studio
Podaj liczbe podzialow przedzialu (np. 100000000): 100000000
Podaj liczbe watkow (np. 4): 4
Przyblizona wartosc liczby Pi: 3.141592653589683
Czas obliczen: 0.5443517000000000 sekund

C:\Users\Uzytkownik\Desktop\P5\Integral\x64\Debug\Integral.exe (proces 13964) zakończono z kodem 0 (0x0).
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Rys. 2.1. działanie

Wyniki pomiarów będą zapisywane w arkuszu kalkulacyjnym, gdzie wiersze będą odpowiadały różnym liczbom wątków, a kolumny czasom wykonania dla różnych liczb podziałów całki. Tabela pozwala na zobrazowanie wpływu liczby rdzeni pro-

cesora na czas wykonania obliczeń oraz na identyfikację, przy jakiej liczbie wątków program osiąga najmniejszy czas wykonania.

## 2.4. Testy algorytmu

Testy algorytmu zostaną przeprowadzone dla różnych liczb podziałów całki (np. 100'000'000, 1'000'000'000, 3'000'000'000) oraz dla liczby wątków od 1 do 50. Wszystkie wyniki zostaną zapisane w tabeli, a następnie posłużą do stworzenia wykresu, gdzie oś Y będzie przedstawiać czas wykonania, a oś X liczbę wątków.

Testowanie programu w systemie Windows będzie także obejmowało monitorowanie zajętości procesora, liczby wątków oraz częstotliwości procesora za pomocą menedżera zadań. Możliwe będzie dodanie zrzutów ekranu przedstawiających te parametry w trakcie wykonywania obliczeń. Po wykonaniu testów algorytmu należy zwrócić uwagę na kilka kluczowych kwestii:

- **Spadek czasu wykonania** – Wzrost liczby wątków powinien prowadzić do skrócenia czasu obliczeń, przynajmniej do momentu, w którym program osiągnie granice wydajności sprzętu.
- **Optymalna liczba wątków** – Należy ustalić, przy jakiej liczbie wątków czas pracy programu osiąga minimum. Zbyt duża liczba wątków może spowodować przeciążenie systemu, co może skutkować wydłużeniem czasu wykonania.
- **Zależność od sprzętu** – Testy powinny być przeprowadzone na różnych komputerach, aby sprawdzić, jak parametry sprzętowe, takie jak liczba rdzeni procesora, wpływają na wyniki.

Wnioski z tych testów pozwolą na lepszą optymalizację algorytmu oraz dokładniejsze zrozumienie, jak różne konfiguracje sprzętowe wpływają na czas obliczeń.

## 2.5. Opis algorytmu i programu poprzez ChatGPT

Algorytm użyty w tym programie ma na celu obliczenie przybliżonej wartości liczby Pi za pomocą metody numerycznego całkowania prostokątów. Całkowanie prostokątów jest popularną metodą przybliżania całek, gdzie całkowanie funkcji jest zastępowane sumą prostokątów o szerokości odpowiadającej krokowi całkowania. Aby obliczyć wartość liczby Pi, używa się funkcji:

$$f(x) = \frac{1}{1+x^2}$$



Całkowanie tej funkcji w przedziale od 0 do 1 daje wartość Pi podzieloną przez 4. Stąd, aby uzyskać wartość Pi, wynik całkowania należy pomnożyć przez 4.

### 2.5.1. Program

Program jest napisany w C++ i implementuje klasę `PiCalculator`, która realizuje obliczenia przybliżonej wartości liczby Pi. Obliczenia są przeprowadzane równolegle przy użyciu wielu wątków, co przyspiesza proces obliczeniowy, szczególnie przy dużej liczbie przedziałów. Kroki działania programu:

- **Konstruktor klasy `PiCalculator`:** Konstruktor przyjmuje dwie wartości:
  - `intervals`: Liczba przedziałów, na które dzielona jest jednostkowa przestrzeń (przedział od 0 do 1), w celu przybliżenia całki.
  - `threads`: Liczba wątków, które zostaną użyte do równoległego obliczania wartości całki.
- **Funkcja `calculate()`:** Główna funkcja odpowiedzialna za obliczenia. Działa w następujący sposób:
  - Podziela całkowanie na mniejsze segmenty, które będą obliczane równolegle przez różne wątków.
  - Dla każdego segmentu (określanego przez zmienne `start` i `end`) wątek oblicza część sumy za pomocą funkcji `computeSegment()`.
  - Po obliczeniu wyników przez wszystkie wątki, wyniki są sumowane.
  - Ostateczny wynik jest mnożony przez 4, aby uzyskać przybliżenie liczby Pi.
- **Funkcja `computeSegment()`:** Oblicza wkład do całki dla określonego segmentu. Iteruje przez wskazany zakres (od `start` do `end`), oblicza wartość funkcji dla każdego punktu w tym zakresie, a wynik sumuje. Po zakończeniu całkowania wynik jest skalowany przez szerokość kroku, aby uzyskać odpowiednią wartość przybliżoną.
- **Funkcja `worker()`:** Funkcja robocza, którą każdy wątek wywołuje, aby obliczyć wkład w sumę w swoim zakresie. Funkcja ta wywołuje `computeSegment()` i zapisuje wynik do zmiennej przekazywanej przez referencję.
- **Pomiar czasu:** Program mierzy czas wykonania obliczeń, aby ocenić wydajność algorytmu. Wykorzystuje bibliotekę `chrono`, która umożliwia dokładne mierzenie czasu przed i po zakończeniu obliczeń.

### **2.5.2. Optymalizacja równoległa**

Dzięki równoległemu przetwarzaniu obliczeń, program jest znacznie szybszy, szczególnie gdy liczba przedziałów jest bardzo duża. Program dzieli pracę na mniejsze segmenty i przypisuje je do różnych wątków. Taki podział pozwala na efektywne wykorzystanie wielordzeniowych procesorów, co skutkuje przyspieszeniem całego procesu obliczeniowego.

### **2.5.3. Podsumowanie**

Algorytm wykorzystuje metodę numerycznego całkowania prostokątów do obliczenia wartości liczby Pi, przy czym obliczenia są równoległe realizowane przez wiele wątków. Dzięki temu program jest szybki i efektywny, szczególnie przy dużych zestawach danych.

## 3. Projektowanie

### 3.1. Wykorzystywane narzędzia

#### 3.1.1. C++

C++ to wydajny język programowania ogólnego przeznaczenia, który rozszerza język C o programowanie obiektowe, szablony oraz mechanizmy umożliwiające kontrolę nad zarządzaniem pamięcią. Język ten jest szeroko stosowany w tworzeniu oprogramowania systemowego, aplikacji desktopowych, gier komputerowych, a także w aplikacjach wymagających dużej wydajności, takich jak oprogramowanie do analizy danych czy symulacje numeryczne. C++ umożliwia programistom bezpośrednią manipulację pamięcią, co daje im pełną kontrolę nad działaniem aplikacji, ale również stawia wyzwania w zakresie zarządzania pamięcią i zapobiegania błędom, takim jak wycieki pamięci. Dzięki mechanizmowi szablonów, C++ umożliwia tworzenie generatywnych algorytmów i struktur danych, które mogą działać na różnych typach danych.<sup>2</sup>

#### 3.1.2. Visual Studio

Visual Studio to jedno z najpopularniejszych zintegrowanych środowisk programistycznych (IDE) firmy Microsoft, które wspiera wiele języków programowania, w tym C++, C, F, Python i wiele innych. To rozbudowane IDE oferuje zaawansowane narzędzia do edycji, debugowania, profilowania i testowania aplikacji. Visual Studio umożliwia programistom łatwe zarządzanie projektami, integrowanie systemów kontroli wersji, a także oferuje różne dodatki, takie jak IntelliSense – funkcję podpowiedzi kodu, która wspomaga pisanie kodu, oraz wsparcie dla systemów takich jak Azure. Visual Studio pozwala na tworzenie aplikacji przeznaczonych na różne platformy – od aplikacji desktopowych, przez aplikacje webowe, po mobilne. Ponadto, IDE to wspiera również rozwój oprogramowania związanego z bazami danych, a także ułatwia tworzenie aplikacji chmurowych.<sup>3</sup>

#### 3.1.3. Biblioteka „thread”

Biblioteka `<thread>` w C++ jest standardową biblioteką, która umożliwia programowanie równoległe w języku C++. Dzięki tej bibliotece, programiści mogą uruchamiać wiele wątków jednocześnie, co znacząco poprawia wydajność aplikacji w

---

<sup>2</sup>C++ Reference [2]

<sup>3</sup>Visual Studio [3]

przypadku zadań wymagających dużych obliczeń. Programowanie wielowątkowe jest szczególnie przydatne w aplikacjach, które muszą przetwarzać dane w czasie rzeczywistym lub w aplikacjach naukowych, które wykonują obliczenia numeryczne. `<thread>` pozwala na łatwe tworzenie, synchronizowanie i zarządzanie wątkami, dzięki czemu umożliwia zrównoleglenie algorytmów. Istnieje również możliwość przekazywania danych pomiędzy wątkami oraz synchronizowania dostępu do wspólnych zasobów, aby uniknąć problemów z wyścigami danych. <sup>4</sup>

#### 3.1.4. Excel

Excel to arkusz kalkulacyjny oferujący szeroką funkcjonalność do obliczeń matematycznych, statystycznych i analitycznych. Używany w wielu branżach do przechowywania danych, analizy oraz tworzenia raportów. Dzięki funkcjom takim jak tabele przestawne, formuły, makra czy wykresy, Excel staje się niezastąpionym narzędziem dla inżynierów, analityków danych, finansistów i wielu innych. W kontekście tego projektu, Excel jest wykorzystywany do zapisania wyników testów algorytmu, takich jak czasy obliczeń przy różnej liczbie wątków, a także do wygenerowania wykresów porównujących te czasy. <sup>5</sup>

#### 3.1.5. Git

Git to system kontroli wersji, który umożliwia śledzenie zmian w plikach oraz współpracę nad kodem źródłowym w ramach zespołów programistycznych. Git pozwala na zarządzanie historią wersji projektu, umożliwiając cofanie zmian, tworzenie nowych gałęzi, oraz integrację różnych wersji kodu. Jest to narzędzie, które znacząco ułatwia pracę zespołową, pozwalając na równoczesną pracę wielu programistów nad tym samym projektem. Dzięki systemowi gałęzi Git pozwala na izolowanie zmian i testowanie nowych funkcji bez wpływania na główną wersję kodu. Ponadto, Git umożliwia tworzenie rozgałęzionych workflowów, co jest szczególnie użyteczne w większych projektach, gdzie programiści pracują nad różnymi funkcjami jednocześnie. Git pozwala również na łatwe rozwiązywanie konfliktów w kodzie, które mogą wystąpić, gdy dwaj programiści dokonają zmian w tym samym fragmencie kodu. Jego wszechstronność oraz możliwość pracy offline sprawiają, że jest jednym z najczęściej wykorzystywanych systemów kontroli wersji na świecie. <sup>6</sup>

---

<sup>4</sup>C++ Thread Library [4]

<sup>5</sup>Microsoft Excel [5]

<sup>6</sup>Git Official Site [6]

### 3.1.6. GitHub

GitHub to platforma hostingowa dla repozytoriów Git, która umożliwia programistom przechowywanie, udostępnianie i współpracowanie nad kodem źródłowym w chmurze. GitHub oferuje szereg funkcji, które ułatwiają współpracę nad projektem, takich jak pull requests, issues, oraz system przeglądu kodu. Dzięki GitHubowi programiści mogą łatwo dzielić się kodem z innymi, prowadzić projekty open-source oraz śledzić postęp prac w czasie rzeczywistym. Platforma umożliwia również integrację z różnymi narzędziami do automatyzacji procesów (np. GitHub Actions), co pozwala na automatyczne testowanie kodu, jego kompilację czy wdrażanie na serwery produkcyjne. GitHub jest także świetnym narzędziem do przechowywania dokumentacji projektów, co sprawia, że jest szeroko stosowany w profesjonalnym świecie IT. GitHub wspiera również rozwój społeczności open-source, umożliwiając programistom tworzenie repozytoriów, które są dostępne dla innych użytkowników na całym świecie. Dodatkowo, GitHub oferuje możliwość integracji z różnymi narzędziami do ciągłej integracji (CI/CD), co sprawia, że proces wytwarzania oprogramowania staje się bardziej zautomatyzowany i efektywny. <sup>7</sup>

### 3.1.7. ChatGPT

ChatGPT to model językowy opracowany przez OpenAI, który wykorzystuje zaawansowane algorytmy sztucznej inteligencji do generowania tekstu w odpowiedzi na zapytania użytkowników. Model ten jest w stanie zrozumieć i generować odpowiedzi na pytania w szerokim zakresie tematów, co czyni go niezwykle użytecznym w różnych dziedzinach, w tym w programowaniu. ChatGPT może być wykorzystywany do automatycznego generowania kodu, wyjaśniania koncepcji programistycznych, a także jako narzędzie wspomagające naukę i rozwiązywanie problemów technicznych. Dzięki swojej zdolności do rozumienia kontekstu i generowania kodu, ChatGPT jest szczególnie użyteczny w projektach programistycznych, gdzie można go wykorzystać do szybkiego prototypowania, rozwiązywania problemów oraz generowania dokumentacji technicznej. W połączeniu z narzędziami takimi jak GitHub, ChatGPT może również wspomagać zautomatyzowane procesy wytwarzania oprogramowania, takie jak przeglądanie kodu czy optymalizacja algorytmów. Model ten jest także pomocny w generowaniu automatycznych testów oraz pomocy przy rozwiązywaniu problemów związanych z algorytmami i strukturami danych, co może skrócić czas rozwijania projektu. <sup>8</sup>

---

<sup>7</sup>GitHub [7]

<sup>8</sup>ChatGPT [8]

## 4. Implementacja

```
1 PiCalculator::PiCalculator(long long intervals, int threads)
2     : intervals_(intervals), threads_(threads), executionTime_(0) {
3 }
```

**Listing 1.** Konstruktor klasy PiCalculator

• **Listing 1.** Konstruktor klasy PiCalculator inicjalizuje obiekt przyjmując dwie wartości: liczbę przedziałów (`intervals`) i liczbę wątków (`threads`), które będą używane do obliczeń. Konstruktor ustawia wartości zmiennych członkowskich `intervals_`, `threads_` oraz `executionTime_` na odpowiednie wartości. Zmienna `executionTime_` jest ustawiana na 0, aby przechować czas wykonania obliczeń, który zostanie obliczony po zakończeniu operacji.

### 4.1. Obliczanie wartości liczby Pi

```
1 double PiCalculator::calculate() {
2     results_.resize(threads_);
3     auto start_time = std::chrono::high_resolution_clock::now();
4
5     std::vector<std::thread> workers;
6     long long segment_size = intervals_ / threads_;
7     for (int i = 0; i < threads_; ++i) {
8         long long start = i * segment_size;
9         long long end = (i == threads_ - 1) ? intervals_ : (i + 1)
10            * segment_size;
11         workers.emplace_back(worker, this, start, end, std::ref(
12            results_[i]));
13     }
14
15     for (auto& thread : workers) {
16         thread.join();
17     }
18
19     auto end_time = std::chrono::high_resolution_clock::now();
20     std::chrono::duration<double> elapsed = end_time - start_time;
21     executionTime_ = elapsed.count();
22
23     double total_sum = 0.0;
24     for (double partial : results_) {
25         total_sum += partial;
26     }
```

```

26     return total_sum * 4.0;
27 }

```

**Listing 2.** Metoda calculate

• **Listing 2.** Metoda calculate odpowiada za obliczanie przybliżonej wartości liczby Pi metodą numerycznego całkowania prostokątów. Obliczenia są podzielone na segmenty, a każdy segment jest obliczany równolegle przez osobny wątek. Najpierw tworzone są wątki, które wykonują obliczenia dla odpowiednich przedziałów. Po zakończeniu obliczeń przez wątki, wyniki są sumowane, a czas wykonania obliczeń jest mierzony i zapisywany do zmiennej `executionTime\`. Zwrócona wartość to przybliżona liczba Pi, obliczona jako suma wszystkich wyników pomnożona przez 4 (zgodnie z wzorem na obliczenie Pi).

## 4.2. Obliczanie segmentu całki

```

1 double PiCalculator::computeSegment(long long start, long long end)
2 {
3     double sum = 0.0;
4     double step = 1.0 / intervals_;
5     for (long long i = start; i < end; ++i) {
6         double x = (i + 0.5) * step;
7         sum += 1.0 / (1.0 + x * x);
8     }
9     return sum * step;

```

**Listing 3.** Metoda computeSegment

• **Listing 3.** Metoda computeSegment oblicza wartość segmentu całki dla danego przedziału `start` i `end`. Obliczenia polegają na sumowaniu wartości funkcji  $\frac{1}{1+x^2}$  w punktach wzdłuż przedziału, przy czym w każdym przypadku dodawana jest połówka szerokości przedziału (`step`). Całkowity wynik segmentu zwracany jest po przeskalowaniu przez `step`, aby uzyskać poprawny wynik numeryczny.

## 4.3. Funkcja robocza do obliczeń

```

1 void PiCalculator::worker(PiCalculator* calculator, long long start
2     , long long end, double& result) {
3     result = calculator->computeSegment(start, end);

```

**Listing 4.** Funkcja robocza worker

• **Listing 4.** Funkcja `worker` jest funkcją roboczą wywoływaną przez każdy wątek. Odpowiada za obliczanie segmentu całki w wyznaczonym przedziale `start` do `end`. Funkcja ta wywołuje metodę `computeSegment`, aby uzyskać wynik obliczeń, który następnie zapisuje do zmiennej `result`. Funkcja ta jest wykorzystywana do równoległego obliczania wartości liczby Pi w ramach metod `calculate`.

#### 4.4. Czas wykonania obliczeń

```
1 double PiCalculator::getExecutionTime() const {  
2     return executionTime_;  
3 }
```

**Listing 5.** Metoda `getExecutionTime`

• **Listing 5.** Metoda `getExecutionTime` zwraca czas wykonania obliczeń w sekundach. Jest to wartość przechowywana w zmiennej `executionTime_`, która jest ustawiana po zakończeniu operacji obliczeniowych w metodzie `calculate`.

#### 4.5. Analiza wyników i wpływ liczby wątków na czas wykonania

W celu zbadania efektywności programu z użyciem różnych ilości wątków, przeprowadziliśmy testy obliczeń liczby Pi dla różnych ilości interwałów (100 milionów, 1 miliard i 3 miliardy) i zmieniającej się liczby wątków (od 1 do 50). Celem było sprawdzenie, czy zwiększenie liczby wątków wpływa na czas obliczeń, co może sugerować poprawność implementacji równoległości.

Wyniki dla różnych ilości wątków i interwałów przedstawiają się następująco:

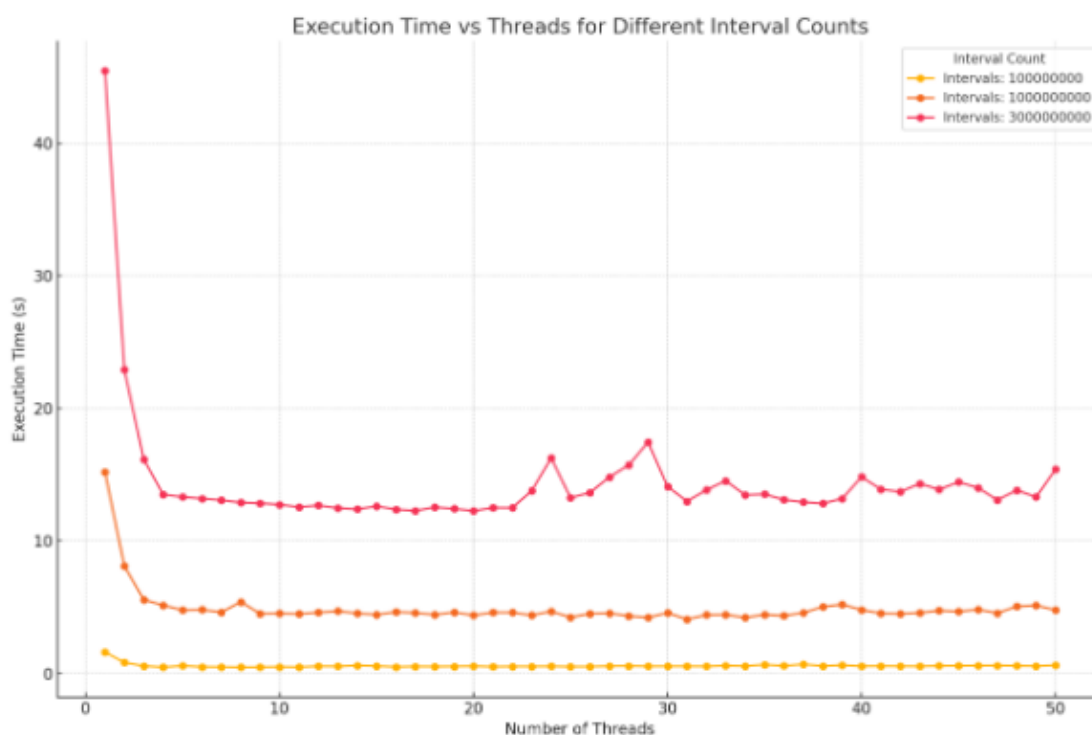
- **100 milionów interwałów:** Zwiększanie liczby wątków początkowo przyspieszało obliczenia, osiągając najniższy czas przy 8 wątkach (0.4707 sekundy). Jednak przy większej liczbie wątków czas wykonania zaczął się zwiększać, co może sugerować, że po pewnym momencie dalsze dodawanie wątków prowadzi do obciążenia systemu lub nadmiernej synchronizacji.
- **1 miliard interwałów:** Dla tej liczby interwałów, czas obliczeń znacząco wzrastał w porównaniu do mniejszej liczby interwałów. Najniższy czas wystąpił przy 1 wątku (15.1967 sekundy), a potem czas ten stopniowo malał do około 4.5 sekundy przy większej liczbie wątków, z wyraźnymi wahaniami w przypadku większej liczby wątków (np. przy 10 wątkach czas wynosił 4.53 sekundy).



- **3 miliardy interwałów:** Dla tej liczby interwałów czas wykonania był znacznie wyższy. Mimo to, czas obliczeń zmniejszał się do około 12.0-13.0 sekundy przy użyciu wielu wątków, ale wzrastał w przypadku dalszego zwiększania liczby wątków, co wskazuje na możliwość wystąpienia przeciążenia.

Wyniki wskazują, że zwiększenie liczby wątków początkowo prowadziło do poprawy wydajności, jednak po pewnym punkcie dalsze zwiększanie liczby wątków może nie przynosić korzyści, a wręcz pogarszać wyniki. Jest to typowe zjawisko związane z nadmierną synchronizacją lub obciążeniem systemu, szczególnie gdy liczba wątków przekracza liczbę dostępnych rdzeni CPU.

Wykres obrazujący czas wykonania obliczeń w zależności od liczby wątków można znaleźć 4.1:



**Rys. 4.1.** Czas wykonania obliczeń w zależności od liczby wątków dla różnych ilości interwałów.

Analizując wyniki, możemy stwierdzić, że parametry sprzętowe (np. liczba rdzeni procesora) mają duży wpływ na czas wykonania programu. Testowanie na różnych komputerach mogłoby ujawnić, w jakim stopniu specyfika hardware'u wpływa na efektywność równoległych obliczeń. Warto również zauważyć, że dla dużych wartości interwałów (np. 1 miliard lub 3 miliardy), czas obliczeń jest zdecydowanie większy, co pokazuje, jak mocno skomplikowane obliczenia numeryczne mogą wpływać na wydajność programu.

## 4.6. Przebieg procesu tworzenia programu

Celem tego projektu było stworzenie programu wielowątkowego, który wykorzystuje algorytm obliczania liczby  $\pi$  metodą całkowania numerycznego. W tym procesie, korzystamy z pomocy czatu AI, aby rozwiązać ewentualne problemy z kodowaniem, testowaniem i optymalizacją programu.

### 4.6.1. Generowanie kodu

Na początku, kluczową kwestią było napisanie poprawnego kodu algorytmu. Został on zaprojektowany z myślą o wielowątkowości, aby zrównoleglić obliczenia. Kod generowano i analizowano wielokrotnie w celu poprawy jego struktury i wydajności. Podczas pisania kodu, czat AI pomógł w odpowiedzi na pytania merytoryczne dotyczące poprawności kodu, jego składni i algorytmów.

### 4.6.2. Kompletność kodu i kompilacja

Po każdej modyfikacji kodu przeprowadzano kompilację programu. Czat AI został użyty do diagnozowania problemów, nie było żadnych błędów i kod działał poprawnie. Istotnym krokiem było również testowanie kodu pod kątem wycieków pamięci. Czat pomagał w identyfikowaniu nieoptymalnych fragmentów kodu, które mogły prowadzić do nieoczekiwanych zachowań w czasie działania programu.

### 4.6.3. Wykrywanie błędów

W przypadku wykrycia błędów w trakcie kompilacji, czat AI sugerował możliwe przyczyny problemów, takie jak brakujące biblioteki, błędy w składni lub niewłaściwie użyte funkcje. Dodatkowo, po uruchomieniu programu, jeśli wyniki były niepoprawne, czat AI pomagał znaleźć potencjalne błędy logiczne i proponował rozwiązania, które mogłyby poprawić wyniki.

### 4.6.4. Testowanie i poprawność wyników

Podczas testowania programu kluczowe było upewnienie się, że uzyskane wyniki są poprawne. Jeśli wyniki nie zgadzały się z oczekiwaniami, czat AI sugerował możliwe przyczyny błędów, takie jak błędne zaokrąglanie wartości, błędy w obliczeniach numerycznych, czy też niewłaściwie zaimplementowane algorytmy równoległe. Dzięki takim podpowiedziom udało się zoptymalizować algorytm i poprawić dokładność obliczeń.

#### 4.6.5. Podsumowanie procesu

Podczas tworzenia programu wielowątkowego z użyciem czata AI, istotnym krokiem było nie tylko napisanie samego kodu, ale także jego testowanie, optymalizacja oraz rozwiązywanie napotkanych problemów. Czat AI był nieocenioną pomocą w wykrywaniu błędów, zarówno kompilacyjnych, jak i logicznych, oraz w optymalizacji kodu. Dzięki temu program działał sprawnie, a jego wyniki były poprawne, co potwierdzono podczas testów.

## 5. Wnioski

### 5.1. Algorytm obliczania liczby Pi

- Algorytm obliczania przybliżonej wartości liczby Pi metodą numerycznego całkowania prostokątów okazał się skutecznym podejściem do obliczeń matematycznych. Wykorzystanie metody równoległej pozwala znacząco przyspieszyć obliczenia, szczególnie przy dużych liczbach przedziałów. Wykorzystanie wielu wątków pozwala na równomierne rozdzielenie obciążenia, co w efekcie skraca czas wykonania obliczeń. Jednakże w przypadku małych danych wejściowych, zalety tej metody są mniej zauważalne, a narzut związany z obsługą wątków może minimalnie obniżyć wydajność.
- Wartość liczby Pi uzyskana za pomocą tej metody była bliska rzeczywistej wartości, a dokładność wyniku rośnie wraz z liczbą przedziałów, co potwierdza słuszność zastosowanego podejścia. Ostateczna dokładność obliczeń zależy głównie od liczby przedziałów oraz liczby wątków, które zostały przydzielone do obliczeń.

### 5.2. Zastosowanie wątków i równoległości

- Równoległe wykonywanie obliczeń pozwala na efektywne wykorzystanie zasobów komputerowych, przy czym liczba wątków powinna być odpowiednio dopasowana do liczby rdzeni procesora, aby uzyskać najlepszą wydajność. Użycie wątków znacznie poprawia czas obliczeń, jednakże zbyt duża liczba wątków w stosunku do liczby rdzeni może prowadzić do nieefektywnego zarządzania zasobami i narzutu związanego z synchronizacją.
- Zastosowanie wielu wątków w przypadku dużych danych wejściowych pozwala na uzyskanie przyspieszenia, które jest zauważalne, gdy dane są rozdzielone na dużą liczbę segmentów.

### 5.3. Git i GitHub

- Git stanowi potężne narzędzie do śledzenia zmian w projekcie oraz umożliwia efektywne zarządzanie wersjami kodu. Dzięki wykorzystaniu systemu kontroli wersji możliwe jest cofnięcie się do poprzednich wersji kodu, co jest szczególnie pomocne w przypadku wystąpienia błędów lub konieczności przeanalizowania wcześniejszych etapów projektu.
- GitHub to platforma, która umożliwia współpracę nad projektem w zespole. Dzięki zdalnym repozytoriom można łatwo współdzielić kod i monitorować postępy pracy. Dzięki narzędziom takim jak pull requesty, issues oraz code reviews, GitHub umoż-

liwia współpracę na najwyższym poziomie, co ułatwia rozwój oprogramowania i utrzymanie jego wysokiej jakości. .

## 5.4. Wnioski z wykorzystania AI przy pisaniu programu

Podczas realizacji projektu pojawiły się istotne pytania dotyczące korzystania z AI jako narzędzia wspierającego programowanie. Odpowiedzi na nie pozwalają lepiej zrozumieć zarówno potencjał, jak i ograniczenia sztucznej inteligencji w tym kontekście.

### 5.4.1. Zagrożenia przy korzystaniu z AI

Korzystanie z AI w procesie pisania programów niesie za sobą pewne zagrożenia, które należy uwzględnić:

- **Generowanie nieoptymalnego kodu:** AI może wygenerować kod, który działa, ale nie jest zoptymalizowany pod kątem wydajności lub struktury.
- **Brak pełnego zrozumienia kodu:** Osoby bez doświadczenia w programowaniu mogą polegać na AI bez zrozumienia działania wygenerowanego kodu, co może prowadzić do problemów przy jego modyfikacji.
- **Zaufanie do błędnych wyników:** AI, podobnie jak człowiek, może popełniać błędy, a generowany kod może zawierać subtelne błędy logiczne lub obliczeniowe.
- **Prywatność i bezpieczeństwo:** W przypadku korzystania z AI, która komunikuje się z chmurą, istnieje ryzyko nieświadomego udostępnienia kodu lub danych wrażliwych.

### 5.4.2. Zalety korzystania z AI

Pomimo wspomnianych zagrożeń, AI oferuje wiele korzyści, które czynią ją wartościowym narzędziem w programowaniu:

- **Przyspieszenie procesu programowania:** AI pomaga szybko generować fragmenty kodu, szczególnie w przypadku powtarzalnych zadań.
- **Rozwiązywanie problemów:** AI może wskazać przyczyny błędów w kodzie i zaproponować poprawki, co skraca czas debugowania.

- **Wspieranie nauki:** Dla początkujących programistów AI może pełnić rolę mentora, dostarczając wyjaśnień i przykładów.
- **Wsparcie w dokumentacji:** AI ułatwia tworzenie dokumentacji technicznej, co jest istotne w większych projektach.

#### 5.4.3. Formułowanie poleceń dla AI

Aby AI mogła generować poprawny i użyteczny kod, polecenia muszą być precyzyjnie sformułowane:

- **Jasność i szczegółowość:** Instrukcje muszą jasno określać wymagania, takie jak język programowania, struktura programu i oczekiwane wyniki.
- **Konkretny kontekst:** Podanie kontekstu, np. typu problemu do rozwiązania, ułatwia AI dostosowanie generowanych rozwiązań.
- **Iteracyjność:** AI lepiej działa, gdy użytkownik iteracyjnie udziela informacji zwrotnej, precyzując swoje potrzeby w kolejnych krokach.

#### 5.4.4. Czy AI może zastąpić programistów?

Na chwilę obecną AI nie jest w stanie w pełni zastąpić programistów, szczególnie tych o dużym doświadczeniu. Powody są następujące:

- **Brak intuicji:** AI nie posiada ludzkiej intuicji, która jest niezbędna do zrozumienia kontekstu i podejmowania decyzji projektowych.
- **Złożoność problemów:** W przypadku złożonych projektów, które wymagają interdyscyplinarnej wiedzy, AI może okazać się niewystarczająca.
- **Zależność od poleceń:** Bez odpowiednio sformułowanych poleceń AI nie wygeneruje poprawnego kodu.

Jednak AI może wspierać osoby początkujące, umożliwiając im realizację prostych projektów, nawet bez pełnej znajomości zasad programowania.

#### 5.4.5. Różnica między czatem AI a GitHub Copilot

Podczas pracy zauważono wyraźne różnice między czatem AI a narzędziem GitHub Copilot:

- **Czat AI:** Jest bardziej interaktywny i umożliwia zadawanie pytań, uzyskiwanie wyjaśnień oraz otrzymywanie kontekstowych odpowiedzi. Idealny do nauki, rozwiązywania problemów i generowania kodu od podstaw.
- **GitHub Copilot:** Skupia się na automatycznym uzupełnianiu kodu w czasie rzeczywistym w środowisku IDE. Jest bardziej zorientowany na poprawę wydajności pracy doświadczonych programistów.

## 5.5. Podsumowanie

Projekt potwierdził skuteczność metody numerycznego całkowania prostokątów w obliczaniu przybliżonej wartości liczby Pi. Równoległe przetwarzanie danych przy pomocy wątków okazało się istotnym elementem w przyspieszaniu procesu obliczeniowego. Użycie narzędzi takich jak Git oraz GitHub umożliwiło łatwe zarządzanie kodem oraz współpracę w zespole. Dzięki temu projekt został zakończony pomyślnie, a jego wyniki stanowią solidną podstawę do dalszego rozwoju. Podsumowując, AI, zarówno w formie czatu, jak i narzędzi takich jak Copilot, stanowi cenne wsparcie w procesie programowania. Kluczowe jest jednak, aby użytkownik rozumiał ograniczenia tych narzędzi i potrafił je wykorzystać w sposób świadomy.

## Bibliografia

- [1] *Strona internetowa Wikipedia*. URL: <https://pl.wikipedia.org/wiki/Pi> (term. wiz. 18.12.2024).
- [2] *C++ Reference*. URL: <https://en.cppreference.com/w/> (term. wiz. 18.12.2024).
- [3] *Visual Studio - Oficjalna Strona*. URL: <https://visualstudio.microsoft.com/> (term. wiz. 18.12.2024).
- [4] *C++ `pthread` Biblioteka*. URL: <https://en.cppreference.com/w/cpp/thread> (term. wiz. 18.12.2024).
- [5] *Microsoft Excel - Oficjalna Strona*. URL: <https://www.microsoft.com/en-us/microsoft-365/excel> (term. wiz. 18.12.2024).
- [6] *Strona internetowa Git*. URL: <https://git-scm.com> (term. wiz. 20.10.2024).
- [7] *GitHub - Platforma Repozytoriów Git*. URL: <https://github.com/> (term. wiz. 18.12.2024).
- [8] *ChatGPT - Model Językowy OpenAI*. URL: <https://openai.com/chatgpt> (term. wiz. 18.12.2024).



## Spis rysunków

2.1. działanie . . . . .	7
4.1. Czas wykonania obliczeń w zależności od liczby wątków dla różnych ilości interwałów. . . . .	17

## Spis listingów

1.	Konstruktor klasy PiCalculator . . . . .	14
2.	Metoda calculate . . . . .	14
3.	Metoda computeSegment . . . . .	15
4.	Funkcja robocza worker . . . . .	15
5.	Metoda getExecutionTime . . . . .	16