AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Wyliczenie przybliżonej liczby PI metodą całkowania numerycznego przy pomocy czatu GPT

Autor:
Szymon Wolski

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2025

Spis treści

1. Ogólne określenie wymagań

Projekt polega na stworzeniu programu wielowątkowego, który wyszukuje liczby pierwsze w określonym zakresie, korzystając z metody sprawdzania podzielności. Główne założenia projektu to:

1. Podział zakresu na segmenty: Cały zadany przedział liczb (np. od 2 do 2 miliardów) jest dzielony na mniejsze części, które są przypisywane do różnych wątków. Liczba wątków może być zróżnicowana, od 1 do nawet 50.

2.Pomiar wydajności: Program porównuje czas potrzebny na wykonanie obliczeń dla różnych rozmiarów zakresów oraz liczby wątków. Wyniki są przedstawiane w formie tabel i wykresów, co pozwala zobaczyć, jak liczba wątków wpływa na czas działania.

3. Zebranie wyników w formie tabelarycznej (arkusz kalkulacyjny), a następnie utworzenie wykresów zależności czasu od liczby wątków. Sprawdzenie poprawności działania programu za pomocą testów jednostkowych (np. Google Test).

4. Analiza wydajności sprzętowej: Program monitoruje zużycie zasobów sprzętowych, takich jak obciążenie procesora, liczba aktywnych wątków, taktowanie procesora oraz użycie pamięci, w różnych wariantach uruchomienia.

Aby sprawdzić, czy dana liczba n jest pierwsza, wykorzystano prostą metodę sprawdzania podzielności:

- Jeżeli $n < 2$, to nie jest ona liczbą pierwszą.
- Jeżeli istnieje jakikolwiek dzielnik d w zakresie od 2 do $\sqrt{n}$, taki że $n \bmod d = 0$, wówczas n nie jest liczbą pierwszą.
- W przeciwnym wypadku n jest liczbą pierwszą.

Wielowątkowość osiągnięto poprzez równomierne dzielenie zakresu liczb pomiędzy wątki. Każdy wątek sprawdza przydzielony przedział i zlicza, ile liczb pierwszych się w nim znajduje. Pod koniec obliczeń sumujemy wyniki.

2. Analiza problemu

2.1. Co to algorytm sprawdzania podzielności

Algorytm *sprawdzania podzielności* (*trial division*) jest jedną z najprostszych metod umożliwiających weryfikację, czy dana liczba n jest liczbą pierwszą. Opiera się na następujących krokach:

1. **Sprawdzenie progu minimalnego**: jeżeli $n < 2$, to z definicji nie jest ona liczbą pierwszą i od razu kończymy procedurę.
2. **Poszukiwanie dzielnika w przedziale** $[2, \sqrt{n}]$: dla każdej liczby całkowitej d z przedziału $[2, \lfloor\sqrt{n}\rfloor]$ sprawdzamy, czy dzieli ona liczbę n bez reszty ($n \bmod d = 0$). Jeśli znajdziemy taką liczbę d, to n *nie* jest liczbą pierwszą, co oznacza, że możemy zakończyć dalsze badanie.
3. **Wniosek o pierwszości**: jeśli w powyższym przedziale nie wystąpił żaden dzielnik, wówczas liczba n jest liczbą pierwszą.

Złożoność obliczeniowa tej metody, rozpatrując pojedynczą liczbę n, wynosi w najgorszym przypadku $\mathcal{O}(\sqrt{n})$. Dla bardzo dużych liczb może to być nieefektywne, dlatego w zastosowaniach wymagających szybkich testów pierwszości (np. w kryptografii) korzysta się z bardziej zaawansowanych i szybszych metod, takich jak Sito Eratostenesa, test Millera–Rabina czy test AKS. Niemniej metoda *trial division* jest intuicyjna i często stosowana w celach edukacyjnych do prostych przykładów.

2.2. Zastosowanie

Algorytm *trial division* może być z powodzeniem stosowany w mniejszych lub edukacyjnych projektach, gdzie zależy nam przede wszystkim na prostocie implementacji i zrozumiałości kodu. Przykładowe obszary zastosowań:

- **Proste programy wyznaczające liczby pierwsze:**
  Jeśli zakres badanych liczb nie jest bardzo duży (np. do kilku milionów), algorytm sprawdzania podzielności może wystarczyć do wygenerowania listy liczb pierwszych na potrzeby zadań szkolnych czy akademickich.
- **Weryfikacja poprawności implementacji innych metod:**
  Często *trial division* wykorzystuje się jako punkt odniesienia w testach. Można porównać jego wyniki z wynikami bardziej zaawansowanych algorytmów (np.

Sit Eratostenesa, testu Millera–Rabina), aby upewnić się, że nowa metoda nie zwraca błędnych rezultatów.

- **Aplikacje edukacyjne:**
  Dzięki przejrzystości i niewielkiej liczbie kroków sprawdzanie podzielności jest idealne do przedstawiania zagadnienia liczb pierwszych w projektach dydaktycznych. Pozwala pokazać etapy pętli sprawdzającej kolejne dzielniki aż do $\sqrt{n}$.

- **Początkowe etapy wyszukiwania liczb pierwszych w dużym zakresie:**
  W niektórych algorytmach można użyć *trial division* do odsiania prostych przypadków (np. dzielenie przez liczby pierwsze z małej puli) zanim zastosuje się kosztowniejsze, zaawansowane testy pierwszości do większych liczb.

W przypadku dużych zakresów lub potrzeby szybkiej weryfikacji (np. w zastosowaniach kryptograficznych) metoda *trial division* nie jest wydajna, dlatego najczęściej korzysta się z bardziej efektywnych algorytmów, takich jak:

- **Sito Eratostenesa** – doskonałe do wyznaczania wszystkich liczb pierwszych w określonym przedziale (szczególnie gdy górna granica przedziału jest znana i nieprzesadnie duża).

- **Test Millera–Rabina** – algorytm probabilistyczny, znacznie szybszy od sprawdzania podzielności, często stosowany w praktyce.

- **Test AKS** – deterministyczny test pierwszości o złożoności wielomianowej, choć w praktyce rzadziej wykorzystywany niż testy probabilistyczne.

Pomimo ograniczeń wydajnościowych, metoda *trial division* bywa więc przydatna wszędzie tam, gdzie kluczowa jest łatwość implementacji i przejrzystość idei.

2.3. Sposób działania

Metoda *trial division* opiera się na prostym założeniu, że jeśli liczba n ma dzielnik mniejszy lub równy $\sqrt{n}$, to nie jest liczbą pierwszą. Główne kroki algorytmu przedstawiają się następująco:

1. **Warunek początkowy:**
   Na początku sprawdzamy, czy $n < 2$. Jeżeli tak, to z definicji n nie jest liczbą pierwszą i kończymy działanie algorytmu.

2. **Wyznaczenie górnej granicy sprawdzania:**
   Obliczamy $k = \lfloor \sqrt{n} \rfloor$. Dalsze sprawdzenia wykonujemy w pętli dla d od 2 do k.

3. **Sprawdzanie podzielności:**
   Wewnątrz pętli sprawdzamy, czy $n \bmod d = 0$. Jeśli tak, to oznacza to, że d jest dzielnikiem n, co natychmiast dowodzi, że n *nie* jest liczbą pierwszą.

4. **Decyzja o wyniku:**

   - Jeżeli w pętli znajdziemy jakikolwiek dzielnik d, który dzieli n bez reszty, algorytm zwraca wynik, że n jest liczbą złożoną.
   - Jeżeli żadna z liczb w przedziale $[2, k]$ nie podzieli n bez reszty, uznajemy n za liczbę pierwszą.

Przeanalizujmy to na przykładzie liczby $n = 37$:

- Najpierw obliczamy $\lfloor \sqrt{37} \rfloor = 6$.
- Następnie sprawdzamy kolejne $d \in \{2, 3, 4, 5, 6\}$.
- Ani 2, ani 3, 4, 5 czy 6 nie dzielą 37 bez reszty, więc wniosek jest taki, że 37 jest liczbą pierwszą.

Złożoność czasowa algorytmu w najgorszym przypadku wynosi $\mathcal{O}(\sqrt{n})$, co oznacza, że dla każdej liczby potrzebujemy do $\sqrt{n}$ prób sprawdzenia podzielności. Dlatego właśnie *trial division* jest uznawana za algorytm intuicyjny, ale niezbyt wydajny przy badaniu bardzo dużych liczb.

3. Projektowanie

Chat GPT

3.0.1. Co to jest ChatGPT?

ChatGPT to duży model językowy (z rodziny *Generative Pre-trained Transformer*), opracowany przez firmę OpenAI. Dzięki wykorzystaniu technik głębokiego uczenia maszynowego oraz trenowaniu na ogromnych zbiorach danych pochodzących z Internetu, jest w stanie generować odpowiedzi w wielu językach, w tym po polsku.

3.0.2. zastosowanie

ChatGPT może być wykorzystywany w wielu dziedzinach i scenariuszach. Kilka przykładowych zastosowań to:

- **Tworzenie i redagowanie tekstów** – generowanie artykułów, esejów, opisów produktów czy streszczeń może być wspomagane przez model językowy. Dodatkowo ChatGPT może służyć do korekty i optymalizacji już istniejących treści.
- **Pomoc w programowaniu** – ChatGPT ułatwia tworzenie kodu, wskazuje potencjalne błędy, a także pomaga w wyjaśnianiu trudnych zagadnień dotyczących różnych języków i bibliotek programistycznych.
- **Wsparcie w obsłudze klienta** – w postaci czatbotów w serwisach internetowych lub systemach call center, gdzie jest w stanie odpowiadać na najczęściej zadawane pytania i rozwiązywać typowe problemy klientów.
- **Generowanie pomysłów i inspiracji** – model może służyć do burzy mózgów, sugerowania koncepcji czy tworzenia scenariuszy w obszarach kreatywnych, takich jak pisanie książek, scenariuszy filmowych czy projektów gier.
- **Tłumaczenia językowe** – choć ChatGPT nie jest specjalistycznym tłumaczem, w wielu przypadkach potrafi pomóc w zrozumieniu lub przeformułowaniu tekstu między różnymi językami.
- **Edukacja i nauka** – może pełnić rolę wirtualnego asystenta w rozwiązywaniu zadań, wyjaśnianiu zagadnień i przy nauce języków obcych, prezentując przykłady i wyjaśnienia.

Należy jednak pamiętać, że ChatGPT, jako model statystyczny, może niekiedy popełniać błędy i wymaga weryfikacji udzielanych odpowiedzi, zwłaszcza w krytycznych zastosowaniach zawodowych czy naukowych.

3.0.3. Sposób działania

ChatGPT jest dużym modelem językowym (z rodziny Generative Pre-trained Transformer) stworzonym przez firmę OpenAI. W swojej budowie i działaniu opiera się na kilku kluczowych etapach:

1. **Architektura transformera:**
   Model ChatGPT wykorzystuje *architekturę transformera*, która umożliwia równoległe przetwarzanie sekwencji tekstu. W przeciwieństwie do starszych rozwiązań RNN (ang. *Recurrent Neural Network*), w transformatorach istotną rolę odgrywa mechanizm *self-attention*, pozwalający modelowi wychwytywać zależności między słowami (tokenami) niezależnie od ich wzajemnego położenia w zdaniu.

2. **Proces trenowania:**
   ChatGPT jest *wstępnie wytrenowany* na ogromnych zbiorach danych tekstowych pochodzących z Internetu. W trakcie uczenia model uczy się przewidywać kolejne słowa (tokeny) w sekwencji, co oznacza, że otrzymuje fragment tekstu i próbuje dokończyć go w najbardziej prawdopodobny sposób. Dzięki temu buduje wewnętrzną reprezentację struktur językowych.

3. **Dopasowywanie odpowiedzi**:
   Po zakończeniu wstępnego treningu model przechodzi fazę *fine-tuning* z nadzorem (instrukcjami), co pozwala dostosować go do konkretnych zadań, takich jak konwersacja czy odpowiadanie na pytania. Dodatkowo wykorzystywane są techniki uczenia ze wzmocnieniem z udziałem człowieka (ang. *Reinforcement Learning from Human Feedback*, RLHF), aby zwiększyć trafność i zrozumiałość generowanych odpowiedzi.

4. **Generowanie tekstu (inferencja)**:
   W momencie, gdy użytkownik zadaje pytanie lub wprowadza polecenie, ChatGPT analizuje wprowadzone dane (kontekst) i przewiduje najbardziej prawdopodobną sekwencję słów (tokenów), którą powinien wygenerować jako odpowiedź. Proces ten przebiega *autoregresywnie*, co oznacza, że model dobiera kolejne słowa na podstawie poprzednio wygenerowanych i kontekstu wejściowego.

5. **Rozumienie i kontekst**:
   Dzięki *mechanizmowi self-attention* ChatGPT może odnosić się do wcześniejszych słów czy zdań i uchwycić długodystansowe zależności językowe. W praktyce oznacza to, że może pamiętać treść konwersacji czy dłuższych tekstów i na tej podstawie formułować spójniejsze odpowiedzi.

6. **Ograniczenia**:
   ChatGPT, będąc modelem statystycznym, może czasem generować odpowiedzi błędne lub niepełne. Nie posiada także wiedzy o zdarzeniach, których nie uwzględniono w danych treningowych (np. wydarzenia najnowsze), a jego wiedza jest ograniczona do czasu zakończenia zbierania danych. Nie jest też *świadomy* w ludzkim rozumieniu tego słowa — jego działanie sprowadza się do wykorzystywania wzorców nauczonych w procesie trenowania.

3.0.4. Wykorzystanie w projekcie

Chat GPT został wykorzystany w celu stworzenia gotowego kodu

3.1. c++

3.1.1. Co to c++

C++ to język programowania ogólnego przeznaczenia, który rozszerza język C o mechanizmy programowania obiektowego, a także inne zaawansowane funkcje. Jego ikona znajduje się na rysunku .3.1

3.1.2. zastosowanie

C++ jest używany w wielu dziedzinach, w tym:

Tworzenie systemów operacyjnych: np. fragmenty systemów takich jak Windows czy Linux.

Aplikacje desktopowe i mobilne: edytory tekstu, programy graficzne, oprogramowanie inżynieryjne.

Programowanie gier: ze względu na wydajność i możliwość pracy z grafiką oraz pamięcią.

Systemy wbudowane: np. oprogramowanie dla mikrokontrolerów. Przetwarzanie wysokowydajne

Rys. 3.1. ikona c++

3.1.3. Sposób działania

C++ umożliwia programowanie zarówno proceduralne, jak i obiektowe. Jego cechy obejmują:

Programowanie obiektowe: C++ wprowadza klasy i obiekty, co pozwala na grupowanie danych oraz funkcji w struktury, co ułatwia zarządzanie złożonością kodu.

Dziedziczenie: Możliwość tworzenia nowych klas na podstawie istniejących, co wspiera ponowne używanie kodu.

Polimorfizm: Pozwala funkcjom i obiektom przyjmować różne formy w zależności od kontekstu.

Zarządzanie pamięcią: W C++ mamy pełną kontrolę nad pamięcią, co pozwala na dynamiczne alokowanie i zwalnianie zasobów.[1]

3.2. visual studio

3.2.1. Co to Visual Studio

Visual Studio to zintegrowane środowisko programistyczne stworzone przez firmę Microsoft. Jest to narzędzie używane do tworzenia, testowania i debugowania aplikacji w różnych językach programowania, takich jak C++, C#, Python, JavaScript, i wielu innych. Jego ikona znajduje się na rysunku 3.2

Rys. 3.2. Ikona VS

3.2.2. Zastosowanie

Visual Studio jest używane przez programistów do tworzenia różnych typów oprogramowania:

Aplikacje desktopowe (Windows Forms, WPF),

Aplikacje webowe (ASP.NET, HTML, CSS, JavaScript),

Aplikacje mobilne (Xamarin, .NET MAUI),

Gry (Unity, Unreal Engine),

Usługi chmurowe (Microsoft Azure).

3.2.3. sposób działania

Visual Studio ułatwia tworzenie oprogramowania, oferując szereg funkcji, które wspierają proces tworzenia kodu:

Edytor kodu: Oferuje wsparcie dla składni wielu języków, podświetlanie kodu, autouzupełnianie, a także sugestie podczas pisania.

Kompilator i debugger: Visual Studio pozwala kompilować kod do plików wykonywalnych i automatycznie wykrywać oraz naprawiać błędy w kodzie.

IntelliSense: Zaawansowana funkcja podpowiedzi podczas pisania kodu, która analizuje kod i proponuje odpowiednie funkcje, metody czy zmienne.

Integracja z systemami kontroli wersji: Visual Studio obsługuje Git, co po-

zwala zarządzać kodem, śledzić zmiany i współpracować z zespołem bezpośrednio z poziomu środowiska IDE.

Debugowanie i profilowanie: Narzędzia do śledzenia błędów oraz wydajności kodu, które pomagają analizować i optymalizować aplikacje.

Szablony projektów: Visual Studio oferuje szereg gotowych szablonów projektów, które ułatwiają rozpoczęcie pracy nad aplikacjami webowymi, desktopowymi czy mobilnymi.[2]

3.3. doxygen

3.3.1. Co to doxygen?

Doxygen to narzędzie do automatycznego generowania dokumentacji technicznej z kodu źródłowego. Obsługuje wiele języków programowania, takich jak C, C++, Java, Python, Fortran, PHP i inne. Doxygen skanuje kod źródłowy i na podstawie specjalnych komentarzy generuje szczegółową dokumentację w formacie HTML, PDF, LaTeX, czy RTF. Jego ikona znajduje się na rysunku 3.3

Rys. 3.3. Ikona doxygen

3.3.2. zastosowanie

Doxygen jest szeroko stosowany w projektach programistycznych, aby generować dokumentację bezpośrednio na podstawie kodu. Jest szczególnie popularny w zespołach pracujących z językami C i C++, ale także w innych środowiskach, które wymagają dokumentacji technicznej. Przykładowe zastosowania:

Dokumentacja bibliotek i API: Umożliwia automatyczne tworzenie dokumentacji dla publicznych interfejsów, funkcji, klas, metod i zmiennych.

Projekty open-source: Doxygen jest popularny w projektach open-source, ponieważ pozwala na łatwe udostępnienie dokumentacji użytkownikom i programistom pracującym nad kodem.

Zarządzanie dużymi projektami: Dzięki automatycznemu generowaniu dokumentacji, Doxygen pomaga programistom zrozumieć struktury dużych i złożonych projektów.

3.3.3. sposób działania

Doxygen działa poprzez analizowanie specjalnych komentarzy w kodzie, które są oznaczane za pomocą znaczników podobnych do tych w języku HTML. Po uruchomieniu Doxygen generuje dokumentację na podstawie tych komentarzy oraz samego kodu, opisując struktury takie jak klasy, funkcje, zmienne i ich relacje.[3]

4. Implementacja

4.1. Program

```
#include <iostream>
#include <thread>
#include <vector>
#include <chrono>
#include <cmath>

/**
 * @brief Calculates the partial sum of the integral for a given range.
 *
 * @param start Starting index of the range.
 * @param end Ending index of the range.
 * @param step Step size (width of each interval).
 * @return The calculated partial sum.
 */
double calculate_partial_sum(long long start, long long end, double step) {
    double sum = 0.0;
    for (long long i = start; i < end; ++i) {
        double x = (i + 0.5) * step;
        sum += 4.0 / (1.0 + x * x);
    }
    return sum;
}

/**
 * @brief Worker function for each thread to calculate a partial sum.
 *
 * @param start Starting index of the range.
 * @param end Ending index of the range.
 * @param step Step size (width of each interval).
 * @param result Reference to the variable where the result is stored.
 */
void thread_worker(long long start, long long end, double step, double& result) {
    result = calculate_partial_sum(start, end, step);
}

/**
```

```
37  * @brief Main function to calculate an approximation of PI using
      numerical integration.
38  *
39  * The program uses multi-threading to distribute the computation
      of the integral
40  * across multiple threads. The user specifies the number of
      intervals and threads
41  * as input.
42  *
43  * @return int Exit status of the program.
44  */
45 int main() {
46      long long num_intervals; ///< Number of intervals for the
      integration.
47      int num_threads; ///< Number of threads to use for parallel
      computation.
48
49      // Input: Number of intervals and threads
50      std::cout << "Enter the number of intervals: ";
51      std::cin >> num_intervals;
52      std::cout << "Enter the number of threads: ";
53      std::cin >> num_threads;
54
55      if (num_threads < 1 || num_intervals < 1) {
56          std::cerr << "Number of threads and intervals must be
      greater than 0.\n";
57          return 1;
58      }
59
60      double step = 1.0 / num_intervals; ///< Step size for the
      integration.
61      std::vector<std::thread> threads(num_threads); ///< Vector to
      store threads.
62      std::vector<double> results(num_threads, 0.0); ///< Vector to
      store results from each thread.
63
64      long long chunk_size = num_intervals / num_threads; ///< Size
      of the chunk for each thread.
65
66      // Start measuring time
67      auto start_time = std::chrono::high_resolution_clock::now();
68
69      // Launch threads
70      for (int i = 0; i < num_threads; ++i) {
```

```
71          long long start = i * chunk_size; ///< Start index for this
    thread.
72          long long end = (i == num_threads - 1) ? num_intervals :
   start + chunk_size; ///< End index for this thread.
73          threads[i] = std::thread(thread_worker, start, end, step,
   std::ref(results[i]));
74      }
75
76      // Wait for all threads to finish
77      for (auto& t : threads) {
78          if (t.joinable()) {
79              t.join();
80          }
81      }
82
83      // Combine results from all threads
84      double pi = 0.0;
85      for (const auto& result : results) {
86          pi += result;
87      }
88      pi *= step;
89
90      // Stop measuring time
91      auto end_time = std::chrono::high_resolution_clock::now();
92      std::chrono::duration<double> elapsed = end_time - start_time;
93
94      // Output the results
95      std::cout << "Calculated value of pi: " << pi << "\n";
96      std::cout << "Time taken: " << elapsed.count() << " seconds\n";
97
98      return 0;
99  }
```

Listing 1. Program

4.2. Najważniejsze elemnty kodu

```
    double calculate_partial_sum(long long start, long long end,
   double step) {
    double sum = 0.0;
    for (long long i = start; i < end; ++i) {
        double x = (i + 0.5) * step;
        sum += 4.0 / (1.0 + x * x);
    }
    return sum;
}
```

Listing 2. Funkcja calculate$_p$*artial*$_s$*um*

Funkcja ta oblicza część sumy całkowej dla podanego zakresu start i end oraz stosuje wzór trapezów do przybliżenia wartości liczby Pi.

```
    void thread_worker(long long start, long long end, double step,
    double& result) {
    result = calculate_partial_sum(start, end, step);
}
```

Listing 3. Funkcja thread$_w$*orker*

Na listingu mamy pokazaną funkcję thread worker, która wywołuje calculate partial sum i zapisuje wynik w referencji result.

```
    long long num_intervals;
    int num_threads;

        std::cout << "Enter the number of intervals: ";
        std::cin >> num_intervals;
        std::cout << "Enter the number of threads: ";
        std::cin >> num_threads;

        if (num_threads < 1 || num_intervals < 1) {
        std::cerr << "Number of threads and intervals must be
   greater than 0.\n";
    return 1;
}
```

Listing 4. Logika funkcji main

Na powyższym listingu mamy pokazaną głowną logikę w main, któa pobiera liczbę przedziałów num intervals or num threads oraz sprawdza poprawność wprowadzonych danych(czy są większe od zera) Funkcja ta bblicza część sumy całkowej dla podanego zakresu start i end oraz stosuje wzór trapezów do przybliżenia wartości liczby Pi.

```
    long long num_intervals;
    int num_threads;

        std::cout << "Enter the number of intervals: ";
        std::cin >> num_intervals;
        std::cout << "Enter the number of threads: ";
        std::cin >> num_threads;

        if (num_threads < 1 || num_intervals < 1) {
        std::cerr << "Number of threads and intervals must be greater than 0.\n";
    return 1;
}
```

Listing 5. Logika funkcji main

Na powyższym listingu mamy pokazaną głowną logikę w main, któa pobiera liczbę przedziałów num intervals or num threads oraz sprawdza poprawność wprowadzonych danych(czy są większe od zera) Funkcja ta bblicza część sumy całkowej dla podanego zakresu start i end oraz stosuje wzór trapezów do przybliżenia wartości liczby Pi.

```
    double step = 1.0 / num_intervals;
    std::vector<std::thread> threads(num_threads);
    std::vector<double> results(num_threads, 0.0);

    long long chunk_size = num_intervals / num_threads;

    for (int i = 0; i < num_threads; ++i) {
        long long start = i * chunk_size;
        long long end = (i == num_threads - 1) ? num_intervals : start + chunk_size;
        threads[i] = std::thread(thread_worker, start, end, step, std::ref(results[i]));
}
```

Listing 6. Tworzenie wątków i podział pracy

Ten fragment kodu oblicza wielkość porcji(chunk size)dla każdego wątku.Również przydziela każdy zakres przedziałów różnym wątkom.

```
    for (auto& t : threads) {
    if (t.joinable()) {
        t.join();
    }
}

    double pi = 0.0;
    for (const auto& result : results) {
    pi += result;
}
pi *= step;
```

Listing 7. Łączenie wątków i sumowanie wyników

ten listing przedstawia join który zapewnia zakończenie wszystkich wątków przed kontynuowaniem programu.Wyniki z każdego wątku są sumowane w zmiennej pi, a następnie mnożone przez step, aby uzyskać końcowy wynik.

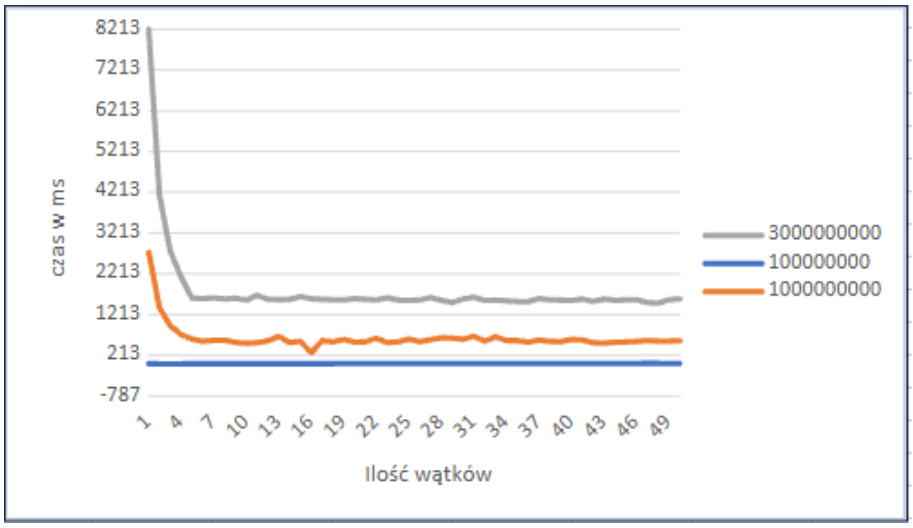
```
    auto start_time = std::chrono::high_resolution_clock::now();
    auto end_time = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = end_time - start_time;

    std::cout << "Calculated value of pi: " << pi << "\n";
    std::cout << "Time taken: " << elapsed.count() << " seconds\n";
```

Listing 8. Pomiar czasu i wyświetlanie wyniku

Powyższy listing przedstawia użycie std:chrono do zmierzenia czasu wykonania obliczeń. Wyświetla przybliżoną wartość liczby pi oraz czas wykonania w sekundach.

4.3. Porównanie wyników

Rysunek 4.1 przedstawia porównanie czasów wykonania w porównaniu do ilości wykorzystanych wątków przy odpowiednich ilościach podziałów (Użyty procesor to ntel(R) Core(TM) i5-8600K CPU @ 3.60GHz 3.60 GHz posiadający 6 rdzeni i 6 wątków).



Rys. 4.1. Porównanie wyników

5. Wnioski

Projekt polegający na implementacji wielowątkowego programu sortującego liczby metodą scalania oraz wyliczania przybliżonej wartości liczby metodą całkowania numerycznego okazał się sukcesem pod wieloma względami.

Implementacja wielowątkowości za pomocą biblioteki std::thread pozwoliła na znaczące skrócenie czasu wykonania operacji sortowania oraz obliczeń całki w porównaniu do wersji jednowątkowej. Testy przeprowadzone z różną liczbą wątków (od 1 do 50) wykazały, że wzrost liczby wątków generalnie przekłada się na redukcję czasu obliczeń. Jednakże, po przekroczeniu liczby rdzeni fizycznych procesora, zaobserwowano spowolnienie wynikające z narzutów związanych z zarządzaniem wątkami oraz przełączaniem kontekstu. Oznacza to, że optymalna liczba wątków jest zazwyczaj równa liczbie dostępnych rdzeni CPU, co maksymalizuje efektywność wykorzystania zasobów sprzętowych.

Chat GPT poradził sobie bez problemu z napisaniem kodu, jednakże AI może generować kod, który wygląda poprawnie, ale zawiera błędy logiczne lub składniowe. Nie zawsze jest w stanie zrozumieć specyficzne wymagania projektu, co może prowadzić do wprowadzenia błędów.

Nadmierne poleganie na AI może prowadzić do braku zrozumienia podstawowych koncepcji programistycznych. Programista może stać się mniej samodzielny w rozwiązywaniu problemów bez wsparcia AI. AI może nieświadomie generować kod podatny na ataki lub zawierający luki bezpieczeństwa, zwłaszcza jeśli jest używany do tworzenia aplikacji o krytycznym znaczeniu.

Istnieją obawy dotyczące praw autorskich w przypadku kodu generowanego przez AI, zwłaszcza jeśli opiera się on na fragmentach istniejących projektów bez odpowiedniego licencjonowania. AI działa na podstawie danych treningowych i może nie mieć pełnego kontekstu projektu, co może prowadzić do nieoptymalnych rozwiązań.

Tak, z pewnymi zastrzeżeniami. AI może znacząco zwiększyć efektywność procesu programowania, zwłaszcza w przypadku rutynowych zadań, szybkiego prototypowania czy generowania szkieletów kodu. Jednak kluczowe jest, aby programista posiadał podstawową wiedzę i umiejętności, aby móc ocenić i poprawić wygenerowany kod. AI powinno być traktowane jako narzędzie wspierające, a nie zastępujące programistę.

Podsumowując chat GPT w dużym stopniu pomógł przy stworzeniu tego projektu oraz był bardzo pomocny, jednakże trzeba być świadomym że stworzone odpowiedzi mogą zawierać błędy więc trzeba sprawdzać co nam ChatGPT wypisuje.

Bibliografia

[1] *Wikipedia c++*. URL: https://pl.wikipedia.org/wiki/C%2B%2B.

[2] *Strona internetowa Visual studio*. URL: https://visualstudio.microsoft.com/pl/.

[3] *Strona internetowa Doxygen'a*. URL: https://doxygen.nl/.

Spis rysunków

Spis listingów