AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Algorytm liczby PI z zastosowaniem GitHub

Autor:
Tomasz Iwański

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań **4**

1.1. Przewidywane wyniki 4

1.1.1. Wyniki merytoryczne 4

1.1.2. Wyniki analityczne dotyczące wydajności 4

1.1.3. Dodatkowe wnioski i analizy 5

1.1.4. Sprawdzanie poprawności działania programu 5

1.2. Zadania związane z dokumentacją 5

1.2.1. Analiza procesu tworzenia kodu 5

1.2.2. Porównanie ChatGPT z GitHub Copilot 6

1.2.3. Dokumentacja techniczna 6

2. Analiza problemu wspomiec o wielowatkowowsci **7**

2.1. Zastosowanie algorytmu całkowania numerycznego 7

2.1.1. Fizyka 7

2.1.2. Inżynieria 7

2.1.3. Ekonomia i finanse 7

2.1.4. Informatyka 7

2.2. Sposób działania programu i algorytmu 7

2.2.1. Idea całkowania numerycznego 7

2.2.2. Proces krok po kroku 8

2.2.3. Równoległość w obliczeniach 8

2.3. Przykład ręcznego obliczenia 8

2.3.1. Podział przedziału [0,1]: 8

2.3.2. Wyznaczenie wartości funkcji w środkach przedziałów 8

2.3.3. Suma prostokątów 9

2.4. Opis narzędzia – ChatGPT 9

2.4.1. Zalety: 9

2.4.2. Wady: 10

2.5. Wykorzystanie narzędzia Git 10

2.6. Dokumentacja projektu 11

3. Projektowanie **12**

3.1. Narzędzia Wykorzystane w Projekcie 12
3.2. Szczegółowe Ustawienia Kompilatora 12
3.3. Schemat Działania Algorytmu 12
3.4. Sposób Używania Narzędzi 13
3.4.1. Git 13
3.4.2. AI 13

4. Implementacja **14**

4.1. Opis Implementacji Algorytmu 14
4.2. Ciekawe Fragmenty Kodu 14
4.3. Powstałe Wyniki 15
4.4. Wnioski z Implementacji 15

5. Wnioski **17**

5.1. Testowanie Algorytmu 17
5.2. Analiza Wydajności 18
5.3. Problemy i Optymalizacje 19
5.4. Zastosowanie Sztucznej Inteligencji w Programowaniu 21
5.5. Przyszłość AI w Programowaniu 21
5.6. Podsumowanie 21

Literatura **22**

Spis rysunków **23**

Spis tabel **24**

Spis listingów **25**

1. Ogólne określenie wymagań

Głównym celem projektu jest stworzenie programu wielowątkowego w języku C++, który oblicza przybliżoną wartość liczby π (PI)[**wikipedia˙Pi**] z wykorzystaniem metody całkowania numerycznego całki oznaczonej. Zadanie to wymaga implementacji równoległości obliczeń za pomocą biblioteki thread, zgodnej z systemami opartymi na standardzie POSIX. Kluczowym aspektem realizacji projektu jest zoptymalizowanie czasu obliczeń poprzez zastosowanie zrównoleglenia, co pozwala na efektywne wykorzystanie współczesnych procesorów wielordzeniowych.

Projekt jest okazją do zrozumienia zarówno teoretycznych podstaw numerycznego obliczania całek, jak i praktycznego zastosowania wielowątkowości w programowaniu współbieżnym. Poza implementacją samego algorytmu, ważne będzie przeprowadzenie gruntownych testów wydajnościowych oraz analiza wyników w zależności od liczby wątków i parametrów sprzętowych.

1.1. Przewidywane wyniki

1.1.1. Wyniki merytoryczne

- Otrzymanie przybliżonej wartości liczby π (PI) z dużą dokładnością, w zależności od liczby podziałów przedziału całkowania. Na przykład, przy podziałach rzędu 100 milionów, 1 miliarda czy 3 miliardów, wartości te powinny konwergować do liczby π z precyzją zależną od rozdzielczości obliczeń.

- Skuteczność metody całkowania numerycznego w kontekście równoległego przetwarzania danych.

1.1.2. Wyniki analityczne dotyczące wydajności

- Tabela czasów: utworzenie tabeli przedstawiającej czasy wykonania obliczeń dla różnych liczb wątków, od 1 do 50, przy zastosowaniu różnych poziomów dokładności (np. 100 mln, 1 mld, 3 mld podziałów).

- Wykresy: wygenerowanie wykresów, na których oś X będzie reprezentować liczbę wątków, a oś Y – czas wykonania programu. Każda linia na wykresie odpowiadać będzie różnej liczbie podziałów przedziału całkowania.

- Analiza skalowalności: zbadanie, w jaki sposób zmiana liczby wątków wpływa na czas wykonania programu. Spodziewanym wynikiem jest zmniejszanie się

czasu obliczeń wraz ze wzrostem liczby wątków, aż do momentu nasycenia, wynikającego z ograniczeń sprzętowych.

1.1.3. Dodatkowe wnioski i analizy

- Minimalny czas obliczeń: identyfikacja liczby wątków, przy której czas obliczeń osiąga minimum. Będzie to punkt, w którym optymalnie wykorzystano dostępne zasoby procesora.

- Porównanie wyników na różnych komputerach: weryfikacja działania programu na różnych konfiguracjach sprzętowych, np. procesorach o różnej liczbie rdzeni i wątków, oraz analiza wyników w kontekście specyfikacji technicznych komputerów.

- Wpływ parametrów sprzętowych: ocena, w jaki sposób takie czynniki jak częstotliwość procesora, liczba rdzeni i wątków logicznych wpływają na efektywność równoległości i czas obliczeń.

1.1.4. Sprawdzanie poprawności działania programu

- Weryfikacja, czy zwiększanie liczby wątków faktycznie redukuje czas działania programu. W przypadku odchylenia od oczekiwanego trendu, konieczna będzie analiza potencjalnych problemów algorytmicznych lub implementacyjnych.

- Kontrola wyników w odniesieniu do poprawności liczbowej – czy obliczone wartości zbliżają się do liczby π w sposób oczekiwany dla różnych poziomów dokładności.

1.2. Zadania związane z dokumentacją

1.2.1. Analiza procesu tworzenia kodu

- Sprawdzenie poprawności generowanego kodu – czy kompiluje się bez błędów, czy działa zgodnie z założeniami, oraz czy generuje wyniki zgodne z oczekiwaniami.

- Dokumentowanie etapów poprawiania kodu, w tym analizowanie, jak generowane są propozycje kodu przez ChatGPT[**wikipedia˙GPT**], i czy proponowane rozwiązania są odpowiednie.

1.2.2. Porównanie ChatGPT z GitHub Copilot

- Wskazanie różnic w użyteczności oraz wad i zalet obu rozwiązań, uwzględniając wcześniejsze doświadczenia z GitHub Copilot[**wikipedia˙Copilot**].

1.2.3. Dokumentacja techniczna

- Przygotowanie pełnej dokumentacji projektu w formacie Latex oraz DoxyGen, z dołączonym kodem źródłowym, opisem wyników oraz analizą wniosków.
- Publikacja projektu na platformie GitHub, w celu umożliwienia łatwego dostępu do kodu oraz dokumentacji.

2. Analiza problemu wspomiec o wielowatkowowsci

2.1. Zastosowanie algorytmu całkowania numerycznego

Algorytmy całkowania numerycznego są powszechnie stosowane w sytuacjach, gdy analityczne obliczenie całki jest trudne lub niemożliwe. Przykłady zastosowań:

2.1.1. Fizyka

- Obliczanie pola pod krzywymi opisującymi ruch ciał, energię czy strumień pola.

2.1.2. Inżynieria

- Symulacje numeryczne systemów dynamicznych.
- Modelowanie przepływów płynów czy obwodów elektrycznych.

2.1.3. Ekonomia i finanse

- Szacowanie obszarów ryzyka w modelach statystycznych.

2.1.4. Informatyka

- Obliczenia związane z grafami, geometryczne przekształcenia, czy numeryczna estymacja wartości stałych matematycznych, takich jak liczb jak π.

W kontekście projektu, metoda całkowania numerycznego pozwala przybliżyć wartość liczby π, co może być bazą do testowania wydajności algorytmów oraz rozwoju narzędzi wielowątkowych.

2.2. Sposób działania programu i algorytmu

2.2.1. Idea całkowania numerycznego

Metoda całkowania numerycznego dzieli przedział [a,b] całkowania na małe odcinki i aproksymuje wartość całki za pomocą prostszych funkcji (np. prostokątów, trapezów). W tym przypadku algorytm obliczy całkę oznaczoną funkcji f(x) na przedziale

[0,1] dla:

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

Formuła ta wynika z geometrycznej definicji liczby π związanej z polem koła.

2.2.2. Proces krok po kroku

- Podział przedziału [0,1] na N równych odcinków.
- Wyznaczenie wartości funkcji f(x) w środkach przedziałów.
- Obliczenie przybliżonej wartości całki jako sumy obszarów prostokątów:

$$\int_a^b f(x)\, dx \approx \sum_{i=1}^{N} f(x_i) \cdot \Delta x, \quad \text{gdzie} \quad \Delta x = \frac{b-a}{N}.$$

2.2.3. Równoległość w obliczeniach

- Przedział całkowania dzielony jest między wątki, które równolegle obliczają wartości funkcji dla przypisanych im podprzedziałów.
- Po zakończeniu obliczeń wyniki z poszczególnych wątków są sumowane, aby uzyskać końcowy wynik.

2.3. Przykład ręcznego obliczenia

Załóżmy, że chcemy przybliżyć wartość liczby dla $\pi = 4$ N=4, gdzie:

$$f(x) = \frac{1}{1+x^2}, \quad a = 0, \quad b = 1.$$

2.3.1. Podział przedziału [0,1]:

$$\Delta x = \frac{1-0}{4} = 0.25.$$

Punkty podziału: $x_0 = 0, \quad x_1 = 0.25, \quad x_2 = 0.5, \quad x_3 = 0.75, \quad x_4 = 1.0.$

2.3.2. Wyznaczenie wartości funkcji w środkach przedziałów

$$x_1' = 0.125, \quad x_2' = 0.375, \quad x_3' = 0.625, \quad x_4' = 0.875.$$

Obliczenia wartości funkcji:

$$f(0.125) = \frac{1}{1+(0.125)^2} = 0.992,$$
$$f(0.375) = \frac{1}{1+(0.375)^2} = 0.933,$$
$$f(0.625) = \frac{1}{1+(0.625)^2} = 0.719,$$
$$f(0.875) = \frac{1}{1+(0.875)^2} = 0.567.$$

2.3.3. Suma prostokątów

$$\text{Całka} \approx \Delta x \cdot \sum_{i=1}^{4} f(x_i') = 0.25 \cdot (0.992+0.933+0.719+0.567) = 0.25 \cdot 3.211 = 0.80275.$$

Przybliżona wartość liczby π:

$$\pi \approx 4 \cdot 0.80275 = 3.211.$$

2.4. Opis narzędzia – ChatGPT

ChatGPT to zaawansowany model sztucznej inteligencji oparty na architekturze GPT-4, który wspiera proces programowania i rozwiązywania problemów. Może być używany do:

- Generowania kodu źródłowego w różnych językach programowania.
- Poprawy istniejącego kodu oraz debugowania.
- Wyjaśniania złożonych algorytmów w sposób przystępny.

2.4.1. Zalety:

- Przyspiesza proces pisania kodu i prototypowania.
- Pomaga rozwiązywać błędy dzięki analizie fragmentów kodu.
- Jest intuicyjny w obsłudze, dostępny dla osób na różnym poziomie zaawansowania.

2.4.2. Wady:

- Wygenerowany kod może wymagać optymalizacji lub poprawek.
- AI nie zawsze rozumie kontekst specyficznych problemów.

Narzędzia takie jak ChatGPT mogą znacząco zautomatyzować procesy programowania, choć ich skuteczność zależy od umiejętności użytkownika w formułowaniu precyzyjnych zapytań oraz weryfikacji wyników.

2.5. Wykorzystanie narzędzia Git

W trakcie realizacji projektu zastosowano system kontroli wersji Git do zarządzania kodem źródłowym oraz synchronizacji z repozytorium na GitHub. Poniżej przedstawiono użyte polecenia wraz z ich opisami:

- git add . - Dodaje wszystkie zmodyfikowane pliki do obszaru staging, przygotowując je do zapisania w następnym commicie.
- it init - Inicjuje nowe lokalne repozytorium Git w bieżącym katalogu. Tworzy ukryty folder .git, który zawiera wszystkie dane potrzebne do zarządzania wersjami
- git commit -m "komentarz Tworzy nowy commit z plików, które zostały wcześniej dodane do obszaru staging, z opisem zmian podanym w "komentarz".
- git log - Wyświetla historię commitów, umożliwiając śledzenie zmian wprowadzonych w projekcie
- git status - Wyświetla stan bieżącego repozytorium, informując o plikach, które zostały zmodyfikowane, dodane lub usunięte, oraz o ich stanie względem obszaru staging.
- git checkout nazwa gałęzi - Przełącza bieżącą gałąź na nazwa gałęzi, umożliwiając pracę nad kodem w innej gałęzi.
- git push - Przesyła lokalne commity do zdalnego repozytorium na GitHub, synchronizując zmiany z innymi użytkownikami.
- git pull - Pobiera najnowsze zmiany z zdalnego repozytorium i synchronizuje je z lokalnym repozytorium, co zapewnia, że mamy aktualną wersję projektu.

- git revert commit hash - Tworzy nowy commit, który odwraca zmiany wprowadzone w określonym commicie oznaczonym commit hash, co jest bardziej bezpiecznym sposobem cofania zmian.
- git rm nazwa pliku - Usuwa plik z obszaru śledzonego przez Git i przygotowuje zmianę do zapisania w następnym commicie.

2.6. Dokumentacja projektu

Cały projekt powinien być udokumentowany za pomocą narzędzia Doxygen, co umożliwi wygenerowanie dokumentacji technicznej dla każdej funkcji i klasy. Dokumentacja ta powinna szczegółowo opisać:

- Algorytm obliczania liczby π: Opis implementacji algorytmu obliczania liczby π metodą całkowania numerycznego, przy użyciu funkcji $f(x) = \frac{1}{1+x^2}$. Zawiera szczegóły dotyczące podziału przedziału całkowania na N podprzedziałów, obliczania wartości funkcji w środkach przedziałów oraz przybliżonego obliczania całki. Złożoność czasowa i przestrzenna algorytmu oraz jego implementacja, w tym zastosowanie wielowątkowości do przyspieszenia obliczeń.
- Proces tworzenia projektu: Wykorzystane technologie i narzędzia, takie jak język C++ oraz biblioteka `thread` do realizacji zrównoleglenia obliczeń. Struktura projektu, organizacja kodu, sposób rozdzielenia odpowiedzialności między pliki nagłówkowe i źródłowe, oraz kluczowe decyzje projektowe, takie jak dobór liczby wątków oraz podział danych na mniejsze fragmenty w celu równoległego przetwarzania.
- Wyniki testów: Szczegółowy opis wyników testów wydajnościowych, które zostały przeprowadzone na algorytmie obliczania liczby π, uwzględniając różne liczby podziałów (np. 100 mln, 1 mld, 3 mld) oraz różną liczbę wątków (od 1 do 50). W analizie wyników testów należy uwzględnić poprawność obliczeń oraz ich wydajność w zależności od liczby wątków, rozmiaru problemu oraz parametrów sprzętowych, takich jak liczba rdzeni CPU.

3. Projektowanie

3.1. Narzędzia Wykorzystane w Projekcie

Do realizacji projektu wykorzystano następujące narzędzia:

- **Język programowania:** C++ [**ww1**]
  Wybrano C++ ze względu na jego wsparcie dla niskopoziomowych operacji, wydajność oraz zaawansowane funkcje wielowątkowości dostępne w bibliotece standardowej.

- **Kompilator:** GCC (Gnu Compiler Collection)[1]
  GCC jest powszechnie używanym kompilatorem kompatybilnym z wieloma systemami operacyjnymi, w tym Linux i Windows.

- **System kontroli wersji:** Git[2]
  Git został użyty do śledzenia zmian w kodzie, zarządzania wersjami i współpracy. Kod został zapisany w repozytorium na platformie GitHub.

- **Biblioteki standardowe:**
  Użyto bibliotek standardowych C++, takich jak `<thread>` do obsługi wielowątkowości, `<mutex>` do synchronizacji wątków oraz `<chrono>` do pomiaru czasu.

- **Latex**[3] **i Doxygen**[4] **:**
  Dokumentacja projektu została przygotowana przy użyciu LaTeX i narzędzia Doxygen do generowania dokumentacji kodu.

3.2. Szczegółowe Ustawienia Kompilatora

Do kompilacji programu użyto następujących flag kompilatora:

- `-std=c++17`: Ustawienie standardu języka C++.
- `-O2`: Optymalizacja kodu dla zwiększenia wydajności.
- `-pthread`: Linkowanie z biblioteką obsługującą wątki POSIX.

3.3. Schemat Działania Algorytmu

Proces działania algorytmu można opisać następująco:

1. Przedział całkowania $[0, 1]$ zostaje podzielony na mniejsze fragmenty w zależności od liczby podziałów określonej przez użytkownika.
2. Fragmenty są rozdzielane pomiędzy dostępne wątki, z których każdy oblicza swoją część sumy całkowej.
3. Wyniki obliczeń z poszczególnych wątków są sumowane w celu uzyskania ostatecznego przybliżenia liczby π.
4. Na koniec program wypisuje czas obliczeń oraz przybliżoną wartość liczby π.

3.4. Sposób Używania Narzędzi

3.4.1. Git

Podstawowe kroki używania Gita w projekcie:

1. Inicjalizacja repozytorium: `git init`
2. Dodanie plików do śledzenia: `git add .`
3. Zapisanie zmian w repozytorium: `git commit -m "Opis zmian"`
4. Wysyłanie kodu na GitHub: `git push origin main`

3.4.2. AI

Podczas tworzenia projektu korzystano z narzędzi AI, takich jak ChatGPT, do:

- Generowania fragmentów kodu.
- Analizy potencjalnych błędów w implementacji.
- Optymalizacji algorytmów i struktury kodu.

AI przyspieszyło proces tworzenia projektu, ale wymagało weryfikacji wyników przez programistę, aby zapewnić poprawność merytoryczną i zgodność z wymaganiami.

4. Implementacja

4.1. Opis Implementacji Algorytmu

Program został zaimplementowany w języku C++ z wykorzystaniem wielowątkowości obsługiwanej przez bibliotekę `<thread>`. Algorytm przybliża wartość liczby π metodą numerycznego całkowania funkcji $f(x) = \frac{4}{1+x^2}$ na przedziale $[0, 1]$.

Całkowity zakres całkowania jest dzielony na fragmenty, które są równomiernie rozdzielane między dostępne wątki. Każdy wątek oblicza swoją część sumy, a następnie wyniki są agregowane, aby uzyskać ostateczną wartość przybliżoną liczby π.

4.2. Ciekawe Fragmenty Kodu

- **Podział zakresu na wątki:** Kod odpowiadający za podział zakresu całkowania na fragmenty przydzielane poszczególnym wątkom jest przedstawiony w Listingu 1.

```
1     double zakres_na_watek = (gorna_granica - dolna_granica) /
      liczba_watkow;
2     for (size_t i = 0; i < liczba_watkow; ++i) {
3         double start_watku = dolna_granica + i *
      zakres_na_watek;
4         double koniec_watku = start_watku + zakres_na_watek;
5
6         watki.emplace_back(oblicz_sume_czesciowa, start_watku,
      koniec_watku, krok,
7                            std::ref(wyniki_czesciowe[i]));
8     }
9
```

Listing 1. Podział zakresu całkowania na wątki.

- **Funkcja do obliczania częściowej sumy:** Obliczenia prowadzone przez każdy wątek są realizowane za pomocą funkcji `oblicz_sume_czesciowa`, przedstawionej w Listingu 2.

```
1     void oblicz_sume_czesciowa(double poczatek, double koniec,
      double krok, double &wynik) {
2         double suma_czesciowa = 0.0;
3         for (double x = poczatek; x < koniec; x += krok) {
4             suma_czesciowa += 4.0 / (1.0 + x * x) * krok;
5         }
```

```
6         wynik = suma_czesciowa;
7     }
8
```

Listing 2. Funkcja obliczająca sumę częściową dla danego zakresu.

- **Pomiar czasu działania algorytmu:** Aby ocenić wydajność, czas trwania obliczeń jest mierzony za pomocą funkcji standardowej biblioteki `<chrono>`. Kod mierzący czas działania przedstawia Listing 3.

```
1     auto czas_start = std::chrono::high_resolution_clock::now()
      ;
2     // Obliczenia
3     auto czas_koniec = std::chrono::high_resolution_clock::now
      ();
4     std::chrono::duration<double> czas_trwania = czas_koniec -
      czas_start;
5
```

Listing 3. Pomiar czasu działania algorytmu.

4.3. Powstałe Wyniki

Podczas testowania programu uzyskano następujące wyniki:

- Przybliżona wartość liczby π z wykorzystaniem różnych liczby przedziałów i wątków była zbieżna z rzeczywistą wartością π (3.141592...).
- Czas działania programu zmniejszał się wraz ze wzrostem liczby wątków, co wskazuje na poprawne zrównoleglenie obliczeń.
- Dla większej liczby przedziałów (np. 10^9) uzyskano większą precyzję, ale również większe zapotrzebowanie na zasoby obliczeniowe.

4.4. Wnioski z Implementacji

Implementacja pokazała, że:

- Biblioteka `<thread>` umożliwia efektywne zrównoleglenie obliczeń na systemach wielordzeniowych.
- Synchronizacja wątków za pomocą `std::mutex` pozwala uniknąć problemów związanych z równoczesnym dostępem do danych.

- Pomiar czasu działania umożliwia ocenę wydajności programu i analizę skalowalności w zależności od liczby wątków.

5. Wnioski

W ramach realizacji tego projektu opracowano program w języku C++, którego głównym zadaniem jest obliczenie przybliżonej wartości liczby π z wykorzystaniem metody całkowania numerycznego. Ta metoda jest jedną z popularnych technik przybliżania wartości π poprzez obliczenie całki oznaczonej nad odpowiednią funkcją matematyczną. Program umożliwia użytkownikowi dostosowanie kilku kluczowych parametrów, takich jak liczba przedziałów całkowania oraz liczba wątków, które mają być użyte do przeprowadzenia obliczeń.

Możliwość zmiany liczby przedziałów całkowania pozwala użytkownikowi określić, jak szczegółowe mają być obliczenia – im więcej przedziałów, tym dokładniejszy wynik, ale jednocześnie większy czas obliczeń. Dodatkowo, program daje użytkownikowi kontrolę nad liczbą wątków, które będą równocześnie przetwarzać obliczenia. Dzięki zastosowaniu wielowątkowości, możliwe jest zrównoleglenie procesu obliczeniowego, co znacząco poprawia wydajność programu, zwłaszcza na nowoczesnych procesorach wielordzeniowych. Zrównoleglenie obliczeń sprawia, że proces ten staje się bardziej efektywny, ponieważ poszczególne części obliczeń mogą być wykonywane równocześnie przez różne rdzenie procesora, co skraca całkowity czas potrzebny do uzyskania wyniku.

Projekt ten uwzględnia więc zarówno aspekty teoretyczne związane z metodami numerycznymi, jak i praktyczne zastosowanie technik programowania współbieżnego w celu uzyskania wydajnych obliczeń. W wyniku tego, użytkownik może dostosować parametry programu do dostępnych zasobów sprzętowych, co pozwala na uzyskanie optymalnego balansu między dokładnością obliczeń a czasem ich wykonania.

5.1. Testowanie Algorytmu

Testowanie algorytmu obliczania wartości liczby π za pomocą całkowania numerycznego zostało przeprowadzone przy różnych konfiguracjach liczby przedziałów całkowania oraz liczby wątków. W ramach testów, eksperymenty obejmowały trzy różne liczby przedziałów całkowania, które miały na celu sprawdzenie, jak zmienia się czas wykonania obliczeń w zależności od rozdzielczości całkowania. Wybrano następujące liczby przedziałów:

100’000’000 – liczba przedziałów, która miała na celu sprawdzenie podstawowej wydajności algorytmu przy stosunkowo mniejszej liczbie przedziałów. 1’000’000’000 – liczba przedziałów, która miała symulować obliczenia przy średniej dokładności całkowania, pozwalając na testowanie efektywności algorytmu przy bardziej złożo-

nych obliczeniach. 3'000'000'000 – liczba przedziałów stanowiąca już dużą wartość, której zadaniem było testowanie granicznych możliwości algorytmu oraz jego zachowania przy bardzo dużych danych. Dla każdej z tych konfiguracji wykonano testy przy różnej liczbie wątków, od 1 do 50 wątków. Liczba wątków była zmieniana w celu zbadania, jak wykorzystanie różnych rdzeni procesora wpływa na czas wykonania obliczeń. Dzięki temu testowi można było zaobserwować wpływ zrównoleglenia na wydajność programu oraz sprawdzić, przy jakiej liczbie wątków następuje optymalizacja czasu obliczeń, a przy jakiej nie ma już zauważalnych korzyści z dodawania nowych wątków.

W czasie testów monitorowane były różne parametry, takie jak czas wykonania całkowitych obliczeń, który był rejestrowany dla każdej kombinacji liczby przedziałów i liczby wątków. Danych tych użyto do stworzenia szczegółowego arkusza kalkulacyjnego, w którym zestawiono czasy obliczeń oraz inne istotne wyniki dla każdej konfiguracji.

Na podstawie zgromadzonych danych opracowano wykres, który przedstawia zależność między czasem obliczeń a liczbą wątków. Wykres ten obrazował, jak rośnie lub maleje czas obliczeń wraz ze wzrostem liczby wątków, a także ukazał moment, w którym dalsze zwiększanie liczby wątków nie prowadziło do poprawy wydajności. Wykresy te stanowią ważną część dokumentacji, ponieważ umożliwiają wizualizację wydajności algorytmu w różnych scenariuszach.

Wyniki testów pozwoliły na identyfikację optymalnej liczby wątków dla różnych konfiguracji liczby przedziałów, a także na lepsze zrozumienie, jak systemy wielordzeniowe mogą być wykorzystywane do przyspieszania obliczeń numerycznych.

Zauważono, że czas obliczeń zmniejszał się wraz ze wzrostem liczby wątków do pewnego momentu, po czym dalsze zwiększanie liczby wątków nie prowadziło już do poprawy wydajności. Wynikało to z ograniczeń sprzętowych systemu oraz efektywności samego algorytmu. Optymalna liczba wątków, przy której czas wykonania był najniższy, zależała od liczby przedziałów całkowania i możliwości procesora.

5.2. Analiza Wydajności

Podczas testowania systemu Windows monitorowano obciążenie procesora, wykorzystanie rdzeni, liczbę wątków oraz częstotliwość procesora przy użyciu Menedżera zadań. Obserwowano, że zwiększenie liczby wątków prowadziło do wyższej zajętości rdzeni, ale po przekroczeniu pewnego limitu, dodatkowe wątki nie wpływały już znacząco na czas wykonania, co wskazywało na konieczność dalszej optymalizacji algorytmu pod kątem skalowalności.

Rys. 5.1. Relacja między czasem wykonania a liczbą wątków przy 100 000 000 przedziałach.

Rys. 5.2. Relacja między czasem wykonania a liczbą wątków przy 1000000000 przedziałach..

5.3. Problemy i Optymalizacje

Program obliczający przybliżoną wartość liczby π metodą całkowania numerycznego działał poprawnie, a wyniki obliczeń były zgodne z oczekiwaniami, co oznacza, że implementacja algorytmu została wykonana w sposób prawidłowy. Zgodność wyników z teorią oraz skuteczność obliczeń przy różnych konfiguracjach liczby przedziałów i wątków świadczyły o poprawności działania podstawowych funkcji programu. Niemniej jednak, po przeprowadzeniu szczegółowej analizy wyników testów oraz monitorowania działania programu w czasie rzeczywistym, zauważono kilka kwestii,

Rys. 5.3. Relacja między czasem wykonania a liczbą wątków przy 3000000000 przedziałach.

które mogą wpływać na wydajność systemu przy bardzo dużych liczbach przedziałów i wątków.

Jednym z głównych powodów spowolnienia działania programu przy bardzo dużych danych jest synchronizacja wątków. W przypadku równoległych obliczeń, kiedy liczba wątków i przedziałów rośnie, proces synchronizacji między wątkami staje się coraz bardziej kosztowny. Każdy wątek musi komunikować się z innymi wątkami, aby zsynchronizować swoje wyniki i uniknąć błędów w obliczeniach. W miarę jak liczba wątków wzrasta, czas potrzebny na synchronizację również rośnie, co może prowadzić do opóźnień w obliczeniach i spadku efektywności algorytmu.

Podział danych to kolejny aspekt, który może powodować opóźnienia w działaniu programu. W przypadku bardzo dużych zestawów danych, proces dzielenia danych na mniejsze części, które następnie są przetwarzane przez różne wątki, może wymagać dużej ilości zasobów obliczeniowych. W sytuacji, gdy dane są podzielone na zbyt wiele fragmentów, lub gdy proces podziału danych nie jest zoptymalizowany, może wystąpić nadmierna ilość operacji związanych z przydzielaniem danych wątkom, co również może prowadzić do opóźnień w obliczeniach.

Aby poprawić wydajność programu w takich przypadkach, warto rozważyć zastosowanie bardziej zaawansowanych metod synchronizacji, które mogą zmniejszyć czas potrzebny na komunikację pomiędzy wątkami. Jednym z takich podejść może być zastosowanie blokowania na poziomie wątków (ang. thread-level locking), które minimalizuje potrzebę synchronizacji przy jednoczesnym zapewnieniu, że dane są poprawnie dzielone i przetwarzane przez różne wątki. Inną metodą, która mogłaby

poprawić wydajność, jest użycie tzw. barier synchronizacyjnych, które pozwalają wątkom na synchronizację w wyznaczonych punktach, co może zmniejszyć koszty związane z ciągłą synchronizacją.

5.4. Zastosowanie Sztucznej Inteligencji w Programowaniu

Podczas pracy nad projektem, korzystanie z AI, takich jak ChatGPT, okazało się bardzo pomocne w rozwiązywaniu problemów programistycznych oraz w generowaniu kodu. AI pomaga w szybkim znajdowaniu rozwiązań problemów syntaktycznych i algorytmicznych, co znacznie przyspiesza proces tworzenia programu. Jednakże, przy bardziej złożonych zagadnieniach programistycznych, AI może mieć trudności z proponowaniem efektywnych rozwiązań, które uwzględniają specyfikę sprzętu lub zaawansowane techniki programistyczne.

Z drugiej strony, AI nie zastępuje całkowicie potrzeby zrozumienia algorytmów oraz umiejętności debugowania, ponieważ generowany kod wymaga często optymalizacji i dostosowania do specyficznych warunków. Programowanie z wykorzystaniem AI jest pomocne, ale wymaga również wiedzy na temat podstawowych zasad programowania i umiejętności rozwiązywania problemów.

5.5. Przyszłość AI w Programowaniu

Sztuczna inteligencja staje się coraz bardziej zaawansowana i może w przyszłości zrewolucjonizować proces tworzenia oprogramowania. Zastosowanie AI do generowania kodu i rozwiązywania problemów programistycznych z pewnością przyspieszy rozwój aplikacji. W miarę jak AI stanie się bardziej precyzyjne i zdolne do zrozumienia bardziej skomplikowanych zależności, programowanie stanie się jeszcze bardziej efektywne. Jednakże, podobnie jak w przypadku innych technologii, tak i w AI należy zachować ostrożność, aby unikać błędów wynikających z nadmiernego polegania na automatycznych rozwiązaniach.

5.6. Podsumowanie

Celem projektu było stworzenie programu wielowątkowego w języku C++ do obliczania przybliżonej wartości liczby π za pomocą metody całkowania numerycznego. Program, po przeprowadzeniu testów, wykazał się wysoką efektywnością w zależności od liczby wątków i przedziałów całkowania. W przyszłości planowane jest dalsze usprawnienie algorytmu oraz optymalizacja zarządzania wątkami, aby poprawić skalowalność obliczeń.

Bibliografia

[1] *Gnu Compiler Collection*. URL: https://gcc.gnu.org (term. wiz. 12. 01. 2025).

[2] *GitHub*. URL: https://en.wikipedia.org/wiki/GitHub (term. wiz. 12. 01. 2025).

[3] *Latex*. URL: https://pl.wikipedia.org/wiki/LaTeX (term. wiz. 12. 01. 2025).

[4] *Doxygen*. URL: https://pl.wikipedia.org/wiki/Doxygen (term. wiz. 12. 01. 2025).

Spis rysunków

5.1. Relacja między czasem wykonania a liczbą wątków przy 100 000 000 przedziałach. 19
5.2. Relacja między czasem wykonania a liczbą wątków przy 1000000000 przedziałach.. 19
5.3. Relacja między czasem wykonania a liczbą wątków przy 3000000000 przedziałach. 20

Spis tabel

Spis listingów

1. Podział zakresu całkowania na wątki. 14
2. Funkcja obliczająca sumę częściową dla danego zakresu. 14
3. Pomiar czasu działania algorytmu. 15