

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

Implementacja wyliczania przybliżonej wartości liczby PI metodą całkowania numerycznego w C++ z pomocą AI

Autor:
Mateusz Basiaga

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2025

Spis treści

1. Ogólne określenie wymagań	4
2. Analiza problemu	5
2.1. Zastosowanie algorytmu	5
2.2. Opis działania programu i algorytmu	5
2.3. Przykład zastosowania algorytmu	6
2.4. Opis narzędzi użytych w projekcie	6
2.4.1. CMake i Ninja	6
2.4.2. Doxygen	6
2.4.3. Git i GitHub Actions	6
2.4.4. Komponenty środowiska kompilacji	7
3. Projektowanie	8
3.1. Wykorzystane narzędzia i technologie	8
3.2. Szczegółowe ustawienia kompilatora	8
3.3. Zastosowanie narzędzi AI w projekcie	9
3.3.1. Proces generowania kodu	9
3.3.2. Ocena poprawności generowanego kodu	10
4. Implementacja	13
4.1. Struktura kodu	13
4.2. Implementacja algorytmu	14
4.3. Test działania programu	16
4.4. Testy wydajności programu	16
5. Wnioski	22
5.1. Zagrożenia przy korzystaniu z AI	22
5.2. Czy należy korzystać z AI przy pisaniu programów?	22
5.3. W czym AI pomaga, a w czym przeszkadza?	22
5.4. Jak muszą być definiowane polecenia, aby AI poprawnie wygenerowała kod?	23
5.5. Czy może programować osoba nieznająca się na programowaniu?	23

5.6. Różnica między ChatGPT a GitHub Copilot	23
Literatura	25
Spis rysunków	26
Spis tabel	27
Spis listingów	28

1. Ogólne określenie wymagań

Celem projektu jest stworzenie programu w języku C++, który umożliwi obliczenie przybliżonej wartości liczby π metodą numerycznego całkowania całki oznaczonej. Program powinien spełniać następujące wymagania:

- Możliwość ustawienia przez użytkownika liczby podziałów przedziału całkowania, co wpływa na dokładność obliczeń.
- Obsługa wielowątkowości z możliwością ustawienia liczby wątków w celu zrównoleglenia obliczeń matematycznych.
- Wykorzystanie biblioteki `thread` zgodnej ze standardem POSIX.
- Wyświetlanie wyniku obliczonej całki oraz czasu wykonywania obliczeń na ekranie terminala.

Zakładamy, że wynikiem programu będzie poprawne obliczenie liczby π z dokładnością zależną od liczby podziałów przedziału oraz efektywne wykorzystanie dostępnych zasobów sprzętowych dzięki zastosowaniu wielowątkowości.

Przewidywane wyniki obejmują:

- Skrócenie czasu obliczeń wraz ze wzrostem liczby wątków, co pozwoli zweryfikować poprawność implementacji wielowątkowości.
- Uzyskanie stabilnych wyników obliczeń π niezależnie od liczby wątków, przy założeniu wystarczającej liczby podziałów przedziału całkowania.
- Obserwację wpływu parametrów sprzętowych (liczba rdzeni, częstotliwość procesora) na wydajność programu.

Oczekujemy, że program pozwoli także na przeprowadzenie testów wydajnościowych i analizy wyników, które zostaną przedstawione w postaci wykresów i tabel. Wyniki te posłużą do oceny efektywności zastosowanych rozwiązań wielowątkowych oraz wyciągnięcia wniosków na temat wpływu ilości wątków na czas wykonania programu.

2. Analiza problemu

2.1. Zastosowanie algorytmu

Algorytmy całkowania numerycznego znajdują szerokie zastosowanie w wielu dziedzinach nauki i techniki. Są szczególnie użyteczne w sytuacjach, gdy funkcja nie posiada analitycznego rozwiązania całki lub gdy dostępne są tylko dane dyskretne. Przykłady zastosowań obejmują:

- Obliczanie pól powierzchni pod wykresami funkcji w fizyce i inżynierii.
- Modelowanie zjawisk przyrodniczych, takich jak przepływ ciepła, dynamika płynów czy mechanika kwantowa.
- Analizę danych eksperymentalnych w statystyce i ekonomii.
- Symulacje komputerowe w informatyce i matematyce stosowanej.

W kontekście tego projektu, metoda numerycznego całkowania służy do aproksymacji wartości liczby π na podstawie całki oznaczonej z funkcji $\frac{4}{1+x^2}$ na przedziale $[0, 1]$.

2.2. Opis działania programu i algorytmu

Program realizuje aproksymację liczby π za pomocą numerycznej metody prostokątów. Algorytm podzielony jest na następujące kroki:

1. Podział przedziału całkowania na określoną przez użytkownika liczbę podprzedziałów.
2. Obliczanie wartości funkcji w środku każdego podprzedziału.
3. Sumowanie wyników mnożonych przez szerokość podprzedziału, aby uzyskać przybliżenie całki.
4. Zastosowanie wielowątkowości w celu równoległego obliczania sum częściowych, co przyspiesza obliczenia dla dużej liczby przedziałów.

Wyniki obliczeń oraz czas wykonania są wyświetlane na konsoli oraz zapisywane do pliku CSV.

2.3. Przykład zastosowania algorytmu

Dla ilustracji, załóżmy, że całkujemy funkcję $\frac{4}{1+x^2}$ na przedziale $[0, 1]$ przy użyciu 4 podprzedziałów:

1. Podzielmy przedział $[0, 1]$ na 4 równe części: $[0, 0.25]$, $[0.25, 0.5]$, $[0.5, 0.75]$, $[0.75, 1]$.

2. Obliczmy wartości funkcji w środku każdego przedziału:

$$\begin{aligned}x_1 &= 0.125, & f(x_1) &= \frac{4}{1 + 0.125^2}, \\x_2 &= 0.375, & f(x_2) &= \frac{4}{1 + 0.375^2}, \\x_3 &= 0.625, & f(x_3) &= \frac{4}{1 + 0.625^2}, \\x_4 &= 0.875, & f(x_4) &= \frac{4}{1 + 0.875^2}.\end{aligned}$$

3. Wyznamy całkę jako sumę wartości funkcji pomnożoną przez szerokość przedziału $dx = 0.25$:

$$\text{Całka} \approx dx \cdot (f(x_1) + f(x_2) + f(x_3) + f(x_4)).$$

Przybliżona wartość całki odpowiada aproksymacji liczby π .

2.4. Opis narzędzi użytych w projekcie

2.4.1. CMake i Ninja

CMake został wykorzystany jako generator systemu budowania, ułatwiający zarządzanie zależnościami i konfiguracją projektu. Ninja, będący szybkim systemem budowania, został zastosowany w celu optymalizacji czasu kompilacji.

2.4.2. Doxygen

Doxygen umożliwił automatyczne generowanie dokumentacji kodu źródłowego w formie strony internetowej oraz plików PDF. Dzięki wykorzystaniu motywu Doxygen Awesome, strona dokumentacji zyskała nowoczesny i czytelny wygląd.

2.4.3. Git i GitHub Actions

System kontroli wersji Git zapewnił śledzenie zmian w kodzie oraz ich zarządzanie. GitHub Actions umożliwił automatyczne testowanie kodu, generowanie dokumentacji i tworzenie wydań projektu.

2.4.4. Komponenty środowiska kompilacji

Kompilacja projektu była przeprowadzana z użyciem GCC na systemie Linux oraz MinGW-w64 dla systemu Windows. Dostosowane środowiska MSYS2 (MinGW64/UCRT64) zapewniły kompatybilność z różnymi platformami.

3. Projektowanie

3.1. Wykorzystane narzędzia i technologie

W projekcie wykorzystano następujące narzędzia oraz technologie:

- **Język programowania:** C++17 – zapewnia wysoką wydajność i wsparcie dla programowania wielowątkowego.
- **Kompilator:** GCC (Linux) oraz MinGW-w64 (Windows) – narzędzia do kompilacji kodu na różnych platformach.
- **System budowania:**
 - **CMake** – generator systemów budowania, umożliwiający łatwe zarządzanie konfiguracją kompilacji i zależnościami.
 - **Ninja** – szybki system budowania przyspieszający proces kompilacji.
- **Kontrola wersji:** Git z repozytorium hostowanym na platformie GitHub.
- **Dokumentacja:**
 - **Doxygen** – automatyczne generowanie dokumentacji kodu w formacie HTML i PDF.
 - Motyw Doxygen Awesome – nowoczesny i czytelny wygląd dokumentacji.
- **Wielowątkowość:** Standardowe biblioteki C++ (`<thread>` i `<vector>`) umożliwiające podział zadań na wiele wątków.
- **Dodatkowe biblioteki:** MSYS2 w wersjach MinGW64 i UCRT64, zapewniające zgodność z różnymi platformami.

3.2. Szczegółowe ustawienia kompilatora

Kompilacja projektu została skonfigurowana z uwzględnieniem następujących parametrów:

- Flagi kompilatora:
 - `-O3` – optymalizacja kodu pod kątem wydajności.
 - `-std=c++17` – ustawienie standardu C++17.
 - `-pthread` – wsparcie dla wielowątkowości.

- Powiązania z bibliotekami:
 - Dynamiczne i statyczne linkowanie bibliotek standardowych.
 - Kompatybilność z systemami Linux oraz Windows poprzez MinGW-w64.
- Generowanie dokumentacji:
 - Narzędzie Doxygen skonfigurowane w pliku `Doxyfile`.
 - Automatyzacja procesu generowania za pomocą GitHub Actions.

3.3. Zastosowanie narzędzi AI w projekcie

W projekcie użyto narzędzia AI – **ChatGPT**, które wspierało proces programowania przez generowanie kodu, udzielanie sugestii oraz pomaganie w rozwiązywaniu problemów. Poniżej przedstawiono szczegółową analizę zastosowania AI, odpowiadając na pytania dotyczące jego roli w tworzeniu oprogramowania.

3.3.1. Proces generowania kodu

Kod w projekcie był generowany na podstawie zapytań do ChatGPT, które dotyczyły różnych aspektów programowania. Oto jak przebiegał proces generowania kodu:

- **Tworzenie funkcji** – ChatGPT sugerował implementację funkcji matematycznych (np. obliczanie całki numerycznej) lub pomocniczych (np. formatowanie wyników, podział pracy między wątki).
- **Pomoc w rozwiązywaniu problemów** – ChatGPT pomagał w rozwiązywaniu błędów, takich jak błędna składnia, problemy z pamięcią lub błędy logiczne w algorytmach.
- **Sugestie dotyczące struktur danych** – Często ChatGPT proponował odpowiednie struktury danych do rozwiązania problemu (np. `std::vector` do przechowywania wyników obliczeń w wątkach).

ChatGPT dostarczał także wyjaśnienia na temat używanych konstrukcji, co pozwalało na lepsze zrozumienie tworzonych rozwiązań.

3.3.2. Ocena poprawności generowanego kodu

1. Poprawność merytoryczna

ChatGPT generował kod, który w większości przypadków był poprawny merytorycznie, jednak niektóre zapytania wymagały poprawek, szczególnie w zakresie optymalizacji algorytmów. Przykładowo:

- Kod początkowy generowany przez AI często nie uwzględniał wielowątkowości, co było wymagane w projekcie.
- Pojawiały się również pomysły na nieoptymalne algorytmy (np. zbędne obliczenia, nieefektywne wykorzystanie zasobów).

Mimo to, ChatGPT dostarczał wartościowych wskazówek, które mogły być łatwo dostosowane do specyfiki projektu.

2. Kompilacja kodu

Po każdej modyfikacji kodu generowanego przez ChatGPT, projekt był kompilowany przy użyciu odpowiednich kompilatorów (GCC dla systemu Linux i MinGW-w64 dla systemu Windows). W większości przypadków kod generowany przez ChatGPT kompilował się bez problemów. Jednak zdarzały się błędy takie jak:

- Brakujące deklaracje funkcji lub niepoprawnie zadeklarowane zmienne.
- Problemy z linkowaniem zewnętrznych bibliotek, które były pomijane w wygenerowanym kodzie.

Te problemy były stosunkowo łatwe do zidentyfikowania i naprawienia.

3. Uruchamianie kodu

Kod wygenerowany przez ChatGPT uruchamiał się poprawnie w większości przypadków. Przykładowo:

- Funkcje obliczające całkę były poprawnie zaimplementowane, a wyniki zwracane przez funkcje były zgodne z oczekiwaniami.
- W przypadku problemów, takich jak niepoprawny podział pracy w wątkach, ChatGPT udzielał sugestii, jak poprawić logikę.

Czasami jednak należało dostosować sposób dzielenia pracy na wątki, aby uniknąć błędów synchronizacji lub problemów z wydajnością.

4. Błędy kompilacji

Po wprowadzeniu wygenerowanego kodu do projektu, zauważono drobne błędy kompilacji, takie jak:

- Literówki w nazwach funkcji lub zmiennych.
- Niezgodności w typach danych (np. `int` vs `long long`).
- Nieprawidłowe użycie wskaźników lub referencji w funkcjach wielowątkowych.

Błędy te były stosunkowo łatwe do naprawienia i w większości przypadków nie wymagały dużej interwencji.

5. Błędy wykonania

Po uruchomieniu programu nie wystąpiły żadne rażące błędy wykonania, takie jak wycieki pamięci czy zawieszenia. ChatGPT generował kod z użyciem standardowych mechanizmów C++ (np. `std::vector`, `std::thread`), które zapewniały odpowiednie zarządzanie pamięcią. Jednak w niektórych przypadkach konieczna była optymalizacja obliczeń, aby przy dużej liczbie wątków uniknąć błędów wynikających z niewłaściwego podziału zasobów.

6. Generowanie kodu

Proces generowania kodu z wykorzystaniem ChatGPT był powtarzalny i wymagał kilku iteracji. Początkowo generowany kod wymagał poprawek, zwłaszcza w kwestii podziału pracy w wątkach i synchronizacji wyników. W rezultacie konieczne było kilkakrotne generowanie i modyfikowanie kodu:

- Pierwsze propozycje generowanego kodu były ogólne i nie uwzględniały wielowątkowości.
- Późniejsze iteracje koncentrowały się na poprawkach związanych z optymalizacją obliczeń i wprowadzeniem równoległości.

7. Poprawność dołączonych bibliotek

ChatGPT poprawnie sugerował użycie odpowiednich bibliotek standardowych C++, takich jak `<vector>` i `<thread>`. Jednak w niektórych przypadkach musiałem ręcznie sprawdzić, czy odpowiednie nagłówki były poprawnie załączone i czy wersja kompilatora obsługiwała wymagane funkcje. Należy jednak szczególnie uważać na sugestie pobierania zewnętrznych bibliotek, ponieważ mogą one zawierać niebezpieczny kod. [Więcej na ten temat \[1\]](#)

8. Wyniki obliczeń

Po każdej iteracji testów i wprowadzeniu zmian sugerowanych przez ChatGPT, otrzymywałem poprawne wyniki obliczeń przybliżonej liczby π . Wartości zwrócone przez program były zgodne z oczekiwaniami matematycznymi, a czas obliczeń zmieniał się zgodnie z liczbą wątków i podziałem pracy.

9. Wnioski z zastosowania narzędzi AI

ChatGPT okazał się cennym narzędziem wspomagającym rozwój projektu. Generował poprawny kod, choć czasami wymagał poprawek w kwestiach optymalizacji i szczegółów implementacji. Dodatkowo, pomoc w rozwiązywaniu błędów kompilacji i wykonania umożliwiła szybkie identyfikowanie problemów i dostosowywanie rozwiązań. Choć nie zastępuje pełnej wiedzy programistycznej, ChatGPT stanowi skuteczne wsparcie w procesie programowania, szczególnie w sytuacjach wymagających szybkich odpowiedzi czy rozwiązań.

4. Implementacja

4.1. Struktura kodu

W projekcie, struktura kodu została zaprojektowana w sposób umożliwiający łatwą rozbudowę oraz utrzymanie. Program podzielony jest na kilka głównych modułów, z których każdy odpowiada za określoną część funkcjonalności.

Główne pliki źródłowe W skład projektu wchodzi następujące pliki:

- `main.cpp` – główny plik, który uruchamia program, obsługuje interfejs użytkownika (wybór liczby wątków i liczby podziałów) oraz zarządza działaniem algorytmu.
- `integral_calculator.cpp` – zawiera implementację klasy `IntegralCalculator`, która odpowiada za obliczenia równoległe, czyli podział obliczeń na wątki, obliczanie części całki oraz agregowanie wyników.
- `integral_calculator.hpp` – nagłówek klasy `IntegralCalculator`, w którym zdefiniowane są interfejsy do obliczeń.
- `utils.cpp` – implementacja pomocniczych funkcji, takich jak `formatNumber`, która formatuje liczby zmiennoprzecinkowe w odpowiedni sposób.
- `utils.hpp` – nagłówek dla funkcji pomocniczych.

4.2. Implementacja algorytmu

Algorytm całkowania numerycznego zaimplementowano w klasie `IntegralCalculator`.

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

Listing 1. Funkcja `calculate()`

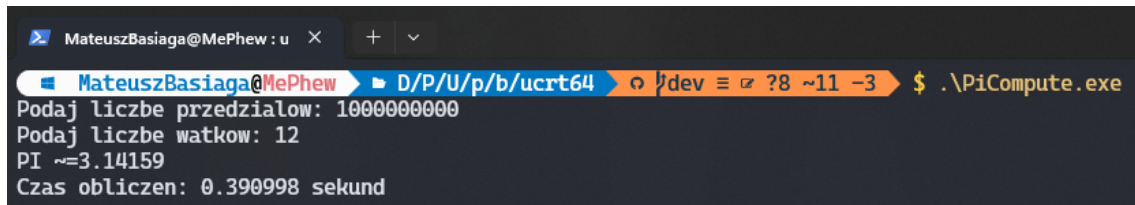
Funkcja `calculate()` klasy `IntegralCalculator` (listing 1 (s. 14)) odpowiada za równoległe obliczenie wartości całki oznaczonej przy użyciu metody prostokątów [2].

Działanie funkcji można podzielić na następujące etapy:

1. **Pomiar czasu rozpoczęcia obliczeń:** Na początku funkcja rejestruje czas rozpoczęcia obliczeń, korzystając z funkcji `std::chrono::high_resolution_clock::now()`, co umożliwia późniejsze obliczenie czasu trwania całego procesu.
2. **Inicjalizacja wątków:** Tworzona jest wektor `threads`, który będzie przechowywał obiekty wątków, oraz wektor `results`, który służy do przechowywania częściowych wyników obliczeń, gdzie każda pozycja odpowiada jednemu wątkowi.
3. **Podział pracy na wątki:** W pętli `for`, iterującej od 0 do `numThreads`, obliczane są zakresy przedziałów (`start` i `end`) dla każdego wątku. Każdy wątek oblicza wartość całki na przypisanym mu fragmencie przedziału, korzystając z funkcji `calculatePartial(start, end)`.
4. **Uruchomienie wątków:** Każdy wątek jest tworzony przy użyciu klasy `std::thread` i dodawany do wektora `threads`. Wątki zapisują wyniki obliczeń w odpowiednim indeksie wektora `results`, dzięki mechanizmowi referencji.
5. **Synchronizacja wątków:** Po uruchomieniu wszystkich wątków, funkcja czeka na ich zakończenie za pomocą metody `join()` wywoływanej dla każdego wątku w wektorze `threads`. Dzięki temu zapewniona jest synchronizacja wyników.
6. **Agregacja wyników:** Po zakończeniu działania wszystkich wątków, w pętli `for` sumowane są wartości przechowywane w wektorze `results`. Ostateczny wynik jest skalowany poprzez przemnożenie przez odwrotność liczby przedziałów (`1.0 / numIntervals`).
7. **Pomiar czasu zakończenia obliczeń:** Rejestrowany jest czas zakończenia obliczeń, a czas trwania całego procesu jest obliczany jako różnica czasu zakończenia i rozpoczęcia, zapisana w zmiennej `computationTime`.

Funkcja jest przykładem zastosowania równoległości w celu przyspieszenia obliczeń numerycznych. Dzięki odpowiedniemu podziałowi pracy na wątki i synchronizacji wyników, możliwe jest wydajne wykorzystanie zasobów procesora.

4.3. Test działania programu



```
MateuszBasiaga@MePheW: u X + v
MateuszBasiaga@MePheW D/P/U/p/b/ucrt64 dev ?8 ~11 -3 $ .\PiCompute.exe
Podaj liczbe przedzialow: 1000000000
Podaj liczbe watkow: 12
PI ~=3.14159
Czas obliczen: 0.390998 sekund
```

Rys. 4.1. Efekt działania programu

Na rysunku 4.1 (s. 16) przedstawiono efekt działania programu dla liczby wątków równej 12 i liczby podziałów równej 1000000000. Jak widać obliczenia zajęły jedynie około 0,39s.

4.4. Testy wydajności programu

```
1 int main(int argc, char* argv[]) {
2     if (argc == 2 && std::string(argv[1]) == "--benchmark") {
3         run_benchmark("benchmark_1", 50, 100'000'000);
4         run_benchmark("benchmark_2", 50, 1'000'000'000);
5         run_benchmark("benchmark_3", 50, 3'000'000'000);
6         run_benchmark("benchmark_4", 50, 10'000'000'000);
7
8         return 0;
9     }
10
11     return show_menu();
12 }
```

Listing 2. Fragment kodu odpowiedzialny za uruchamianie trybu testu wydajności

Na listingu 2 (s. 16) przedstawiono fragment kodu odpowiedzialny za uruchamianie trybu testu wydajności w przypadku, gdy program zostanie uruchomiony z parametrem `--benchmark`.

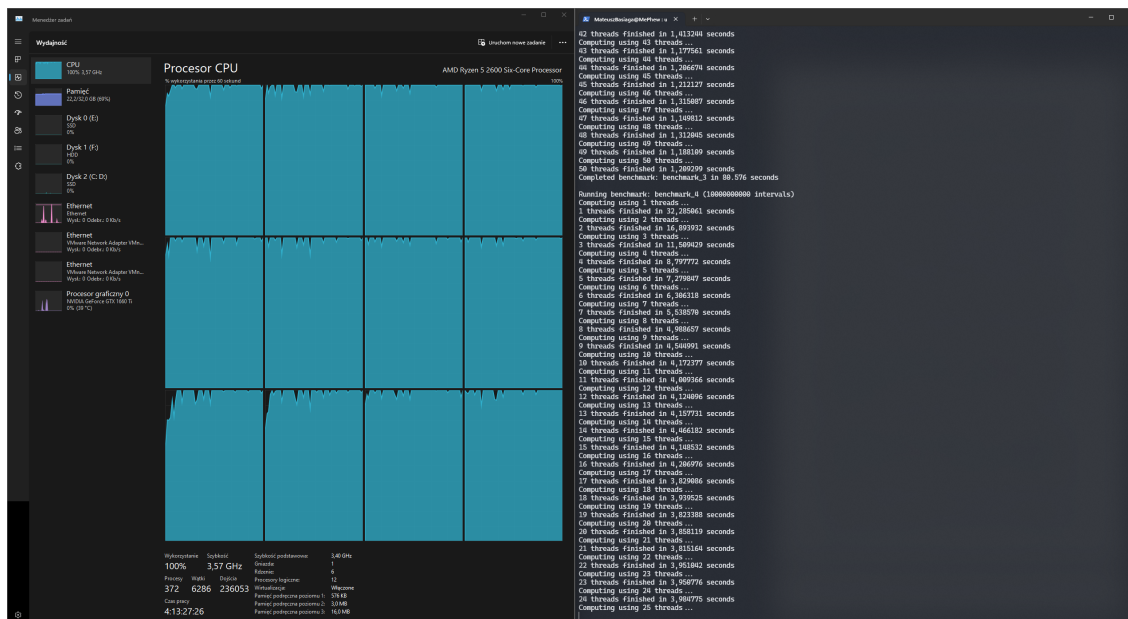

```

1 void run_benchmark(const std::string& name, int max_threads, long
   long max_intervals, int step) {
2     std::ofstream file;
3     file.open(name + ".csv");
4
5     std::cout << "Running benchmark: " << name << " (" <<
       max_intervals << " intervals)" << std::endl;
6
7     double totalTime = 0;
8
9     for (int i = 1; i <= max_threads; i += step) {
10         std::cout << "Computing using " << i << " threads..." << std::
            endl;
11
12         IntegralCalculator integralCalculator(max_intervals, i);
13         integralCalculator.calculate();
14
15         double computationTime = integralCalculator.getComputationTime
            ();
16         totalTime += computationTime;
17
18         std::string formattedResult = formatNumber(computationTime);
19
20         file << i << ", " << formattedResult << std::endl;
21         std::cout << i << " threads finished in " << formattedResult <<
            " seconds" << std::endl;
22     }
23
24     file.close();
25     std::cout << "Completed benchmark: " << name << " in " <<
        totalTime << " seconds" << std::endl;
26     std::cout << std::endl;
27 }

```

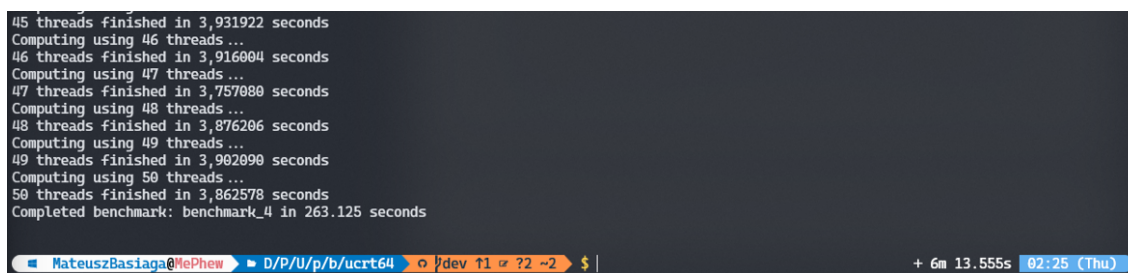
Listing 3. Funkcja runBenchmark odpowiedzialna za wykonanie pojedynczego testu wydajności

Funkcja runBenchmark (listing 3 (s. 17)), uruchamia algorytm dla danej liczby wątków oraz liczby podziałów, a następnie zapisuje wyniki do pliku benchmark.csv. Na ich podstawie sporządzono arkusz programu Excel, pozwalający na szczegółową analizę wydajności programu.



Rys. 4.2. Zużycie procesora podczas obliczeń

Na rysunku 4.2 (s. 18) przedstawiono zużycie procesora podczas wykonywania testu wydajności. Jak widać program jest bardzo zasobożerny.



Rys. 4.3. Zakończenie testu wydajności

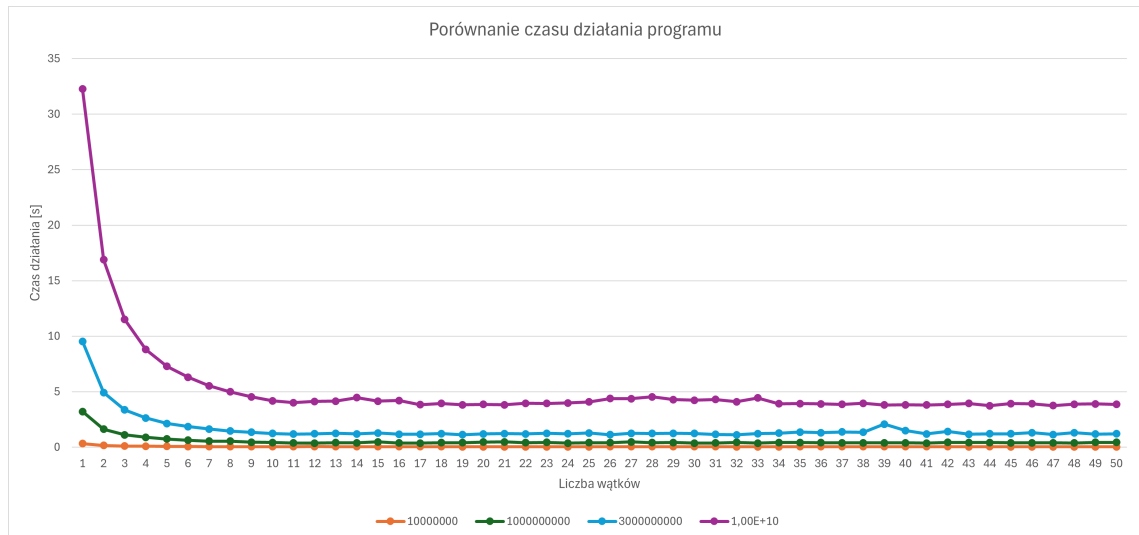
Rysunek 4.3 (s. 18) przedstawia zakończenie testu wydajności. Jak widać przeprowadzenie całego testu zajęło niewiele ponad 263 sekundy (około 4 minuty).

Wątki	10000000	1000000000	3000000000
1	1,01	9,898	30,1
2	0,504	5,012	15,84
3	0,344	3,423	11,123
4	0,262	2,625	8,016
5	0,213	2,114	6,475
6	0,177	1,778	5,488
7	0,154	1,539	4,884
8	0,136	1,353	4,226
9	0,121	1,233	3,696
10	0,11	1,092	3,369
11	0,104	1,003	3,071
12	0,108	0,961	3,179
13	0,12	1,074	3,196
14	0,127	1,067	3,184
15	0,144	1,036	3,12
16	0,137	1,046	3,166
17	0,129	1,037	2,959
18	0,122	1,091	2,94
19	0,117	1,005	3,133
20	0,112	1,048	3,335
21	0,111	1,061	3,214
22	0,103	1,092	3,146
23	0,124	1,046	3,378
24	0,098	1,135	3,025
25	0,125	1,113	3,292
26	0,124	1,127	3,044
27	0,121	1,029	3,253
28	0,12	1,142	3,585
29	0,113	1,016	3,115
30	0,111	1,069	3,263
31	0,108	1,043	3,017
32	0,104	1,025	3,037
33	0,103	1,035	3,235
34	0,1	1,01	3,212
35	0,102	0,998	3,133
36	0,099	0,991	3,19

37	0,119	0,98	3,186
38	0,133	0,979	3,052
39	0,113	0,987	3,154
40	0,11	1,043	3,299
41	0,108	0,98	3,14
42	0,106	1,041	3,111
43	0,103	1,011	3,119
44	0,104	1,01	3,101
45	0,101	1,004	2,981
46	0,098	1,002	3,028
47	0,099	1,026	3,08
48	0,097	1,009	3,007
49	0,112	0,985	2,976
50	0,11	1,003	2,918

Tab. 4.1. Tabela zawierająca czas obliczeń dla poszczególnych wartości liczby podziałów w zależności od liczby wątków

W tabeli 4.1 (s. 20) przedstawiono tabelę z wynikami testów wydajności dla poszczególnych wartości liczby podziałów w zależności od ilości liczby wątków.



Rys. 4.4. Wykres czasu obliczeń dla poszczególnych wartości liczby podziałów w zależności od liczby wątków

Na wykresie (rys. 4.4 (s. 21)) przedstawiono zależność czasu obliczeń od liczby wątków dla poszczególnych wartości liczby podziałów. Różnice możemy zaobserwować przy większych liczbach podziałów, gdzie zwiększenie liczby wątków powoduje znaczne skrócenie czasu obliczeń. Warto zauważyć, że po przekroczeniu liczby 12 wątków czas obliczeń nie ulega już znaczącej zmianie, co może wynikać z ograniczeń sprzętowych (ilość wątków procesora).

5. Wnioski

Podczas realizacji projektu z wykorzystaniem narzędzi opartych na sztucznej inteligencji (AI) można sformułować następujące wnioski:

5.1. Zagrożenia przy korzystaniu z AI

Korzystanie z AI w programowaniu niesie ze sobą pewne ryzyko:

- **Błędy w generowanym kodzie:** Kod generowany przez AI może zawierać błędy logiczne, syntaktyczne lub niezgodności z oczekiwaniami użytkownika, szczególnie gdy polecenia są niedokładne.
- **Niewłaściwe praktyki programistyczne:** AI czasami generuje kod, który nie spełnia standardów bezpieczeństwa, wydajności lub czytelności.
- **Zależność od AI:** Programiści mogą uzależnić się od narzędzi AI, tracąc umiejętność samodzielnego rozwiązywania problemów.
- **Ograniczona kontekstualność:** AI może nie znać pełnego kontekstu projektu, co prowadzi do błędnych założeń i nieoptymalnych rozwiązań.

5.2. Czy należy korzystać z AI przy pisaniu programów?

Korzystanie z AI przy pisaniu programów może być bardzo korzystne, pod warunkiem, że użytkownik jest świadomy jego ograniczeń. Narzędzia AI warto wykorzystywać do:

- Automatyzacji powtarzalnych zadań.
- Szybkiego generowania szablonów kodu.
- Prototypowania rozwiązań i eksploracji różnych podejść.

Jednak należy pamiętać, że AI nie zastępuje wiedzy programistycznej, a jedynie ją uzupełnia.

5.3. W czym AI pomaga, a w czym przeszkadza?

Pomoc:

- Przyspiesza pisanie kodu dzięki generowaniu gotowych fragmentów.

- Pomaga w nauce nowych technologii, sugerując rozwiązania na podstawie najlepszych praktyk.
- Wspiera w identyfikowaniu błędów i ich naprawianiu.

Przeszkody:

- Może generować kod, który jest trudny do zrozumienia lub wymaga znacznych poprawek.
- Wymaga precyzyjnie zdefiniowanych poleceń, aby uniknąć niepożądanych rezultatów.

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

Aby AI generowała poprawny i użyteczny kod, polecenia muszą być:

- **Precyzyjne:** Opisujące dokładnie wymagania oraz oczekiwany efekt.
- **Kontekstowe:** Zawierające informacje o środowisku, języku programowania, używanych bibliotekach itp.
- **Krótkie, ale szczegółowe:** Zawierające konkretne wytyczne, np. „Napisz funkcję w języku C++, która oblicza całkę oznaczoną”.

5.5. Czy może programować osoba nieznająca się na programowaniu?

Na chwilę obecną, osoba bez wiedzy programistycznej może korzystać z AI, aby stworzyć prosty kod. Jednak:

- Brak zrozumienia generowanego kodu może prowadzić do problemów z jego poprawianiem i debugowaniem.
- Bardziej zaawansowane projekty wymagają wiedzy technicznej, aby zrozumieć błędy i poprawnie je naprawić.

5.6. Różnica między ChatGPT a GitHub Copilot

- **ChatGPT:**

- Generuje kod na podstawie naturalnych poleceń tekstowych.
- Nadaje się do wyjaśniania problemów, nauki i szerokich kontekstów programistycznych.
- Może działać poza edytorem kodu, oferując bardziej uniwersalne zastosowanie.

- **GitHub Copilot:**

- Generuje kod w czasie rzeczywistym, bezpośrednio w edytorze.
- Koncentruje się na uzupełnianiu kodu i integracji z istniejącym projektem.
- Jest bardziej zoptymalizowany dla szybkiej pracy w środowisku IDE.

Podsumowując, narzędzia AI, takie jak ChatGPT i GitHub Copilot, są wartościowymi wspomagaczami w procesie programowania, ale wymagają świadomego i krytycznego podejścia do ich wykorzystania.

Bibliografia

- [1] *Portal Dark Reading*. URL: <https://www.darkreading.com/application-security/chatgpt-hallucinations-developers-supply-chain-malware-attacks> (term. wiz. 08.01.2025).
- [2] *Materialy AGH*. URL: https://home.agh.edu.pl/~dpawlus/pliki/matlab/MO_algorytmy.pdf (term. wiz. 08.01.2025).
- [3] *ChatGPT*. URL: <https://chatgpt.com/> (term. wiz. 08.01.2025).
- [4] *Microsoft Excel*. URL: <https://www.microsoft.com/pl-pl/microsoft-365/excel> (term. wiz. 08.01.2025).

Spis rysunków

4.1. Efekt działania programu	16
4.2. Zużycie procesora podczas obliczeń	18
4.3. Zakończenie testu wydajności	18
4.4. Wykres czasu obliczeń dla poszczególnych wartości liczby podziałów w zależności od liczby wątków	21

Spis tabel

- 4.1. Tabela zawierająca czas obliczeń dla poszczególnych wartości liczby
podziałów w zależności od liczby wątków 20

Spis listingów

1. Funkcja `calculate()` 14
2. Fragment kodu odpowiedzialny za uruchamianie trybu testu wydajności 16
3. Funkcja `runBenchark` odpowiedzialna za wykonanie pojedynczego testu wydajności 17