AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Szacowanie liczby pi za pomocą metody całkowania numerycznego całki oznaczonej, algorytm wielowątkowy

Autor:
Mateusz Stanek

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Celem projektu jest utworzenie programu Obliczającego przybliżoną liczbę π metodą całkowania numerycznego całki oznaczonej z funkcji w języku `C++`. Program ma też być wielowątkowy oraz ma być w stanie generować pomiary czasu działania w zależności od liczby wątków. W trakcie pisania należy też skorzystać z chataGPT, w celach pomocnych.

Przewidywany jest działający program, repozytorium git, wyniki benchmarków w postaci pliku i dokumentacja w Doxygenie.

2. Analiza problemu

2.1. Algorytm obliczania π metodą całkowania numerycznego

Algorytm obliczania π metodą całkowania numerycznego (połączenie metody Archimedesa[**archimedes**] i sum Riemanna[**riemannwiki**]) polega na wykorzystaniu wzoru na pole koła

$$P = \pi r^2,$$

Wzór ten można przekształcić do postaci

$$\pi = \frac{P}{r^2},$$

gdzie P to pole koła, a r to promień. Z tej relacji wynika, że można użyć wartości promienia i pola koła do obliczenia π. Jeżeli wiadomo jaki jest promień koła, do określenia pola pod łukiem ćwiartki okręgu i pomnożenia wyniku przez 4 (dlatego, że koło jest symetryczne względem osi x i y układu współrzędnych). Funkcję określająca kształt łuku ćwiartki można uzyskać poprzez przekształcenie wzoru okrąg

$$x^2 + y^2 = r^2,$$

gdzie x, y to koordynaty punktów na okręgu, a r to promień. Przekształcając wzór do postaci $f(x, r) \to y$ uzyskujemy

$$y = \sqrt{r^2 - x^2},$$

gdzie $y, x \geq 0$ (dlatego, że obliczana jest tylko ćwiartka). Pole pod tą funkcją to całka oznaczona

$$P_{cw} = \int_0^r \sqrt{r^2 - x^2}\, dx.$$

Pole całego koła to byłoby zatem

$$P = 4P_{cw}.$$

Należy natomiast zauważyć, że nie da się obliczyć dokładnej wartości tej całki bez wykorzystania liczby π. Dlatego, należy skorzystać z sum Riemanna[**riemannwiki**], aby obliczyć aproksymację wartości π, poprzez podkładanie prostokątów z szerokością o danej wartości i wysokości określonej poprzez wartość funkcji $y = \sqrt{r^2 - x^2}$, gdzie x to punkty oddzielone od siebie o daną szerokość.

Jeżeli jako promień przyjmie się 1, to

$$\pi = \frac{P}{1^2} \implies \pi = P.$$

To założenie upraszcza kalkulacje, ponieważ nie trzeba dzielić pola.

2.2. Git

Kolejnym konceptem, którym zajmuje się projekt jest narzędzie git[2]. Pozwala ono zarządzać poszczególnymi wersjami projektów. Głównym korzeniem gita jest system commitów, czyli zapisania zmian w pliku w stosunku do commita starszego. To, w połączeniu z jego innymi możliwościami pozwala na tworzenie długich i skomplikowanych osi czasu danych projektów.

Użycie gita można zademonstrować na prostym przykładzie. Tworzymy katalog a w nim repozytorium, uzywając komendy `git init`, jak widać na rys. 2.2.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/mattys/skrypty-i-syfy/studia/rok2/progr
amowanie-zaawansowane/p1/git-test/.git/
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % ls
drwxr-xr-x - mattys  6 Nov 15:54  .git/
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test %
```

Rys. 2.1. Puste repozytorium git

Stwórzmy jakiś plik i dodajmy go do repozytorium. Plik można dodać do repozytorium komendą `git add`

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo "commit1" > plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git add plik.txt
```

Rys. 2.2. Stworznie pliku w repozytorium

Następnie należy scommitować zmiany.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git commit -am "utworzenie pl
ik.txt"
[master (root-commit) bb3b898] utworzenie plik.txt
 1 file changed, 1 insertion(+)
 create mode 100644 plik.txt
```

Rys. 2.3. Commit nr. 1

Na rysunku 2.4 użyta komenda `git commit` commituje wszystkie dodane pliki (`-a`) z jakimś komunikatem (`-m`).

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo "commit2" >> plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git commit -am "uzupelnienie
plik.txt"
[master 40694c2] uzupelnienie plik.txt
 1 file changed, 1 insertion(+)
```

Rys. 2.4. Commit nr. 2

Na rys. 2.5, został utworzony kolejny commit, dodajacy zmiany do `plik.txt`.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git log
commit 40694c2e73758368445cb817f4524ee5c5afbece (HEAD → master)
Author: mattys1 <mattys0082@gmail.com>
Date:   Wed Nov 6 16:26:07 2024 +0100

    uzupelnienie plik.txt

commit bb3b898df760395b5763575e204c3c43b513d2f6
Author: mattys1 <mattys0082@gmail.com>
Date:   Wed Nov 6 16:12:31 2024 +0100

    utworzenie plik.txt
```

Rys. 2.5. Log gita

Jak na rys. 2.6 jest pokazane, używając komendy `git log`, można wyświetlić log commitów w repozytorium.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git checkout bb3b898df760395b
5763575e204c3c43b513d2f6
Note: switching to 'bb3b898df760395b5763575e204c3c43b513d2f6'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at bb3b898 utworzenie plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo plik.txt
plik.txt
```

Rys. 2.6. Demonstracja checkout

Jak widać na rys. 2.7, komenda `git checkout`, pozwala na przejście repozytorium w inny stan, w tym przypadku przechodzi się do commita o danym ID, pokazanym na rys. 2.6. Jako, że jest to pierwszy commit, nie ma w nim zmian z drugiego.

2.3. Doxygen

Doxygen[3] jest narzędziem automatycznie generującym dokumentację programu z komentarzy w kodzie źródłowym. Potrafi on generować strony HTML, gdzie można dynamicznie nawigować się miedzy rożnymi częściami kodu oraz pliki LaTeX, które można konwertować na różne, statyczne formaty.

2.4. Github Copilot

GitHub Copilot[4] to model `LLM` oferowany przez GitHub - może on analizować kod źródłowy i funkcjonować jako zaawansowany autocompleter lub asystent potrafiący tworzyć proste fragmenty. Jest on bezpośrednio zintegrowany z wieloma narzędziami Microsoftu, takimi jak Visual Studio Code czy zwykłe Visual Studio. Jednak, jest on dostępny również jako rozszerzenie do innych edytorów, jak Neovim, którego instalacja przy użyciu menagera pluginów `Lazygit` jest ukazana na rys. 2.8.

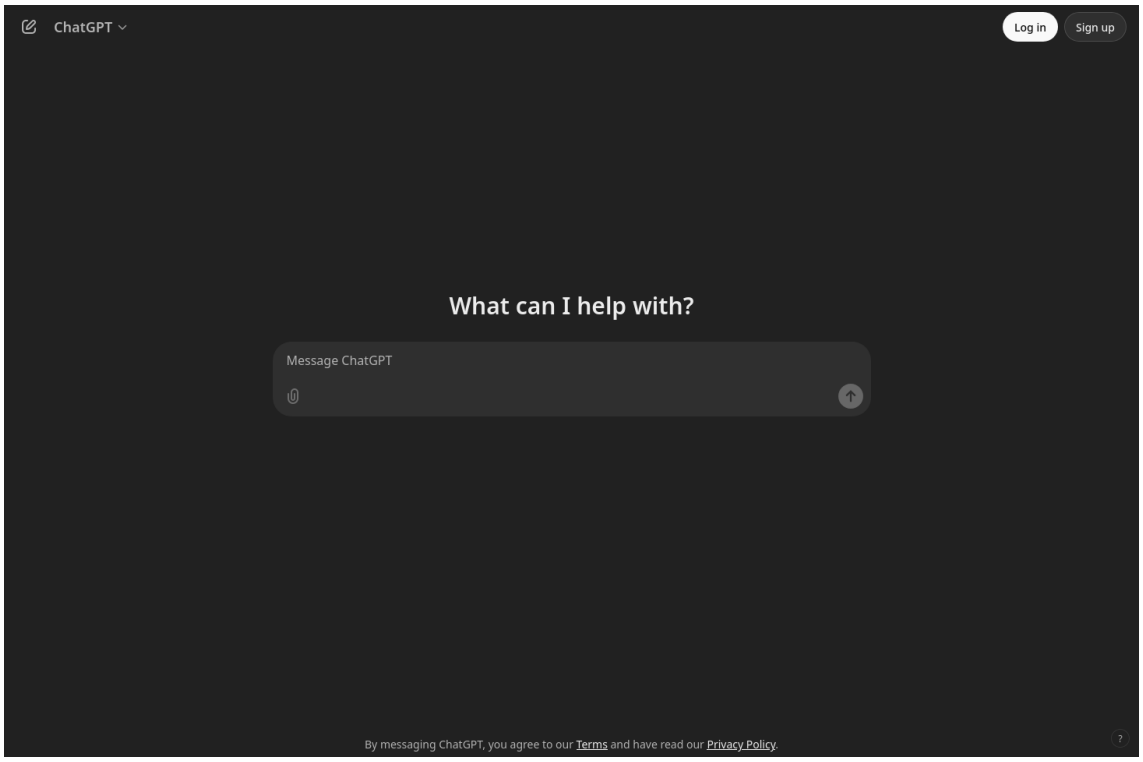
```
return {
    "github/copilot.vim",
    config = function()
        vim.keymap.set('n', '<leader>cp', function ()
            vim.cmd('Copilot')
        end, { noremap = true, silent = true })
    end
}
```

Rys. 2.7. Plik instalacyjny Plugina Copilot

W Visual Studio jest on zainstalowany domyślnie.

2.5. ChatGPT

ChatGPT[**chatgptsite**] jest serią modeli LLM oferowanych przez firmę OpenAI. W przeciwieństwie do Copilota jest on modelem bardziej skoncentrowanym na ogólnej tematyce, aniżeli na programowaniu. Jest on głównie nastawiony na funkcje czatu, ale oferowane jest też API do integracji z innymi narzędziami. Wygląd strony głównej zawarty jest na rysunku nr. **??**.



Rys. 2.8. Strona główna ChatGPT

3. Projektowanie

3.1. Implementacja Algorytmu

Do zaimplementowania Algorytmu obliczania π zostanie użyty Język `C++` z kompilatorem `g++`. Wersja standardu `C++` to `C++23`. Systemem budowy projektu będzie `CMake`, pomagający w konfiguracji wersji kompilatora czy bibliotek. `CMake` pozwala na generowanie plików budujących dany projekt, zgodnie z określoną konfiguracją. Oszczędza to programiście, szczególnie przy większych projektach, manualne pisanie Makefileów. Plik konfiguracyjny `CMakeLists.txt` może wyglądać jak na rysunku

```
cmake_minimum_required(VERSION 3.15)

set(PROJECT_NAME proj4)

project(${PROJECT_NAME} VERSION 0.1 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 23)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_EXPORT_COMPILE_COMMANDS True)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_CURRENT_LIST_DIR}/out)

add_subdirectory(src)

```

Listing 1. Plik konfiguracyjny CMake

Edytorem będzie program Neovim. Jest to terminalowy edytor tekstu z możliwością poszerzenia funkcjonalności przy użyciu wszelkiego rodzaju pluginów. Wybrany został, dlatego że jest on już skonfigurowany na moim komputerze zgodnie z moimi preferencjami.

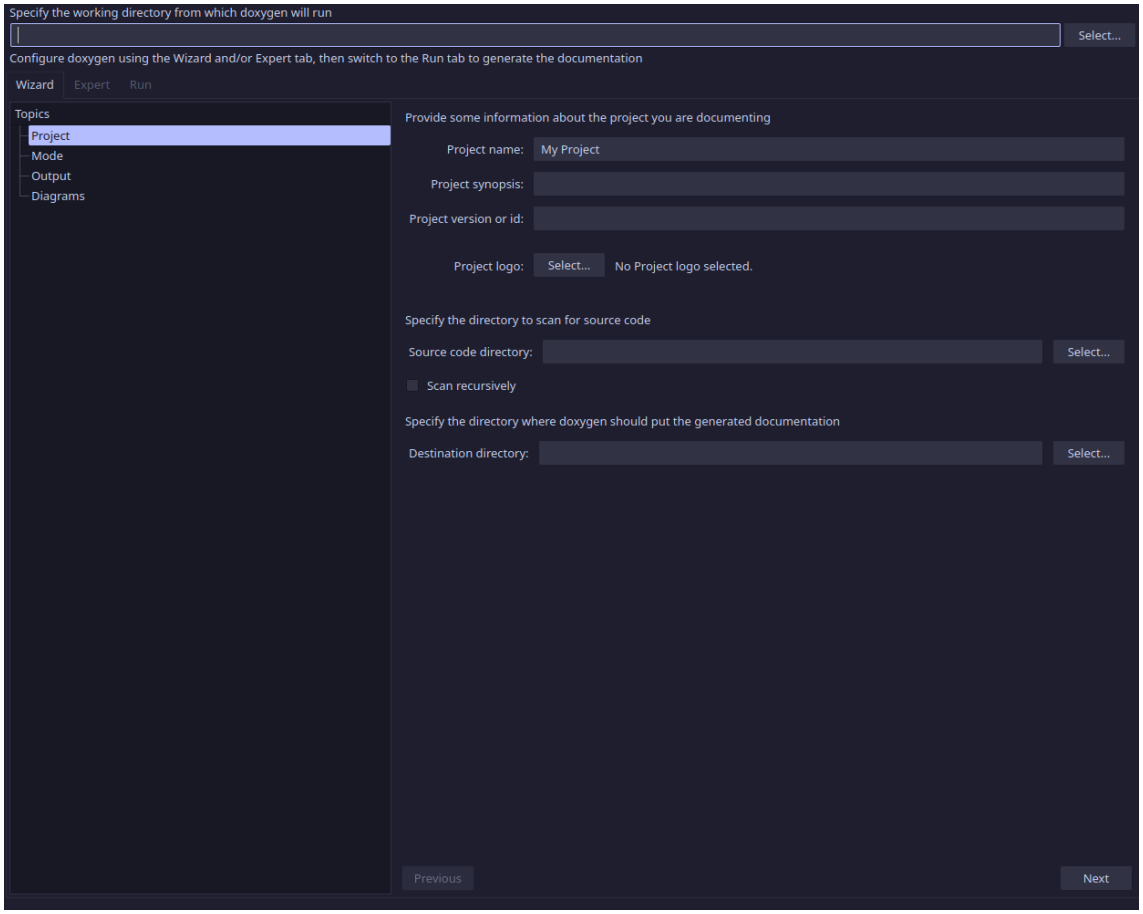
3.2. Git

Dla ułatwienia pracy, zastosowany został front-end dla gita o nazwie lazygit. Jest to terminalowy program, którego główną zaletą jest łatwa nawigacja przy użyciu klawiatury. Ponadto, jest on lekki i szybki.

3.3. Doxygen

Konfiguracja dla Doxygena jest wygenerowana przy użyciu programu doxywizard, pokazany na rys. 3.1, pozwalającego na graficzne zmienianie ustawień. Po wygene-

rowaniu konfiguracji, Doxygen wywoływany jest przy użyciu komendy.



Rys. 3.1. Interfejs programu doxywizard

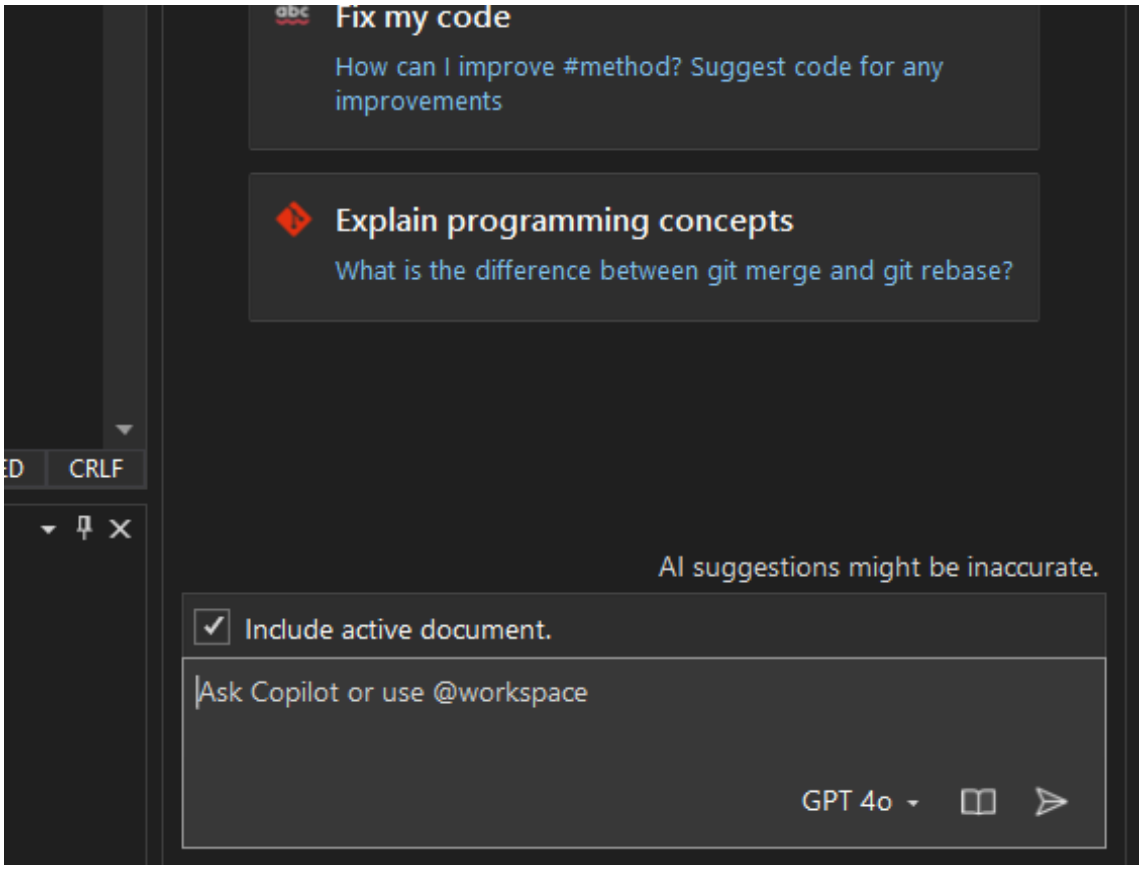
3.4. Github Copilot

Do napisania programu do pomocy został wykorzystany Github Copilot. Z pomocy Copilota można skorzystać na 3 sposoby. Sposób pierwszy - poprzez podpowiedzi generowane przez Copilota które są oznaczone kolorem szarym, co pokazano na rysunku nr 3.2.

```cpp
#include <iostream>

int main()
{
    int choice;
    std::cout << "Choose a number between 1 and 3: ";
    std::cin >> choice;
    if(choice == 1)
    {
        std::cout << "You chose 1" << std::endl;
    }
    else if(choice == 2)
    {
        std::cout << "You chose 2" << std::endl;
    }
    else if(choice == 3)
    {
        std::cout << "You chose 3" << std::endl;
    }
    else
    {
        std::cout << "You chose a number that is not between 1 and 3" << std::endl;
    }
    return 0;
}
```

Rys. 3.2. Sugerowanie przez copilot

Sposób drugi to standardowe okienko chat gdzie można zapytać go o rzeczy związane z kodem, przedstawione zostało na rysunku nr 3.3



Rys. 3.3. okienko chat Copilot

Trzeci sposób to wykorzystanie komentarzy jako poleceń co ma zrobić copilot, jak pokazano na rysunku nr 3.4

```
// Funkcja kalkulatora dodająca, odejmująca, mnożąca i dzieląca dwie liczby
int kalkulator(int a, int b, char dzialanie)
{
    if (dzialanie == '+')
    {
        return a + b;
    }
    else if (dzialanie == '-')
    {
        return a - b;
    }
    else if (dzialanie == '*')
    {
        return a * b;
    }
    else if (dzialanie == '/')
    {
        return a / b;
    }
    else
    {
        return 0;
    }
}
```

Rys. 3.4. Zastosowanie komentarzy do generowania kodu

4. Implementacja

4.1. Implementacja Algorytmu obliczania π

Algorytm zaimplementowany jest w funkcji `getPi()`, pokazanej na listingu nr. ?? zawartej w `main.cpp`.

```
1  double getPi(const size_t steps, const size_t threads) {
2    long double deltax { 1. / steps };
3
4    if(deltax > 1) {
5      std::println("Rectangle width should be smaller than 1");
6      return -1;
7    }
8
9    if(threads == 0) {
10     std::println("Threads should be a non-zero value");
11     return -1;
12   }
13
14   double sum = 0.;
15
16   auto calculate = [&](const long double start, const long double
       end) {
17     long double localSum = 0.;
18     for(long double i {start}; i < end; i+= deltax) {
19       localSum += sqrt(1 - i * i) * deltax;
20     }
21
22     sum += localSum;
23   };
24
25   std::vector<std::thread> threadPool;
26   threadPool.reserve(threads);
27
28   for(int i = 0; i < threads; i++) {
29     threadPool.push_back(std::thread([&, i] {
30       calculate(
31         (1. / static_cast<double>(threads)) * i,
32         (1. / static_cast<double>(threads)) * (i + 1)
33       );
34     }));
35
36   }
37
```

```
38    for(auto &thread : threadPool) {
39      thread.join();
40    }
41
42    return sum * 4;
43  }
```

Listing 2. Funkcja `getPi()`

Na linijce nr. 1 zawarta jest deklaracja funkcji `getPi()`. Parametr `steps` określa gęstość prostokątów w ćwiartce, czyli dokładność obliczeń. Parametr `threads` natomiast, określa liczbę wątków jaka ma być używana podczas działania algorytmu. W linijce nr. 2, parametr `deltax` to szerokość poszczególnego prostokąta. W linijce nr. 4 i nr. 9 sprawdzana jest poprawność parametrów. W linijce nr. 14 zadeklarowana jest zmienna `sum`, która ma przechowywać sumę pól prostokątów. W linijce nr. 16 zadeklarowana jest funkcja lambda `calculate()`, która oblicza sumę pól prostokątów dla danego przedziału. Przedział określony jest przez parametry `start` i `end`. Na końcu lambdy, po zsumowaniu pól zmienna `localSum` dodawana jest do zmiennej `sum` zbieranej przez referencję. Od linijek nr. 25-34 tworzony jest wektor `threadPool` i wypełniany jest on wątkami, które po stworzeniu wywołują lambdę `calculate()`. Na końcu, w linijkach nr. 38-40, czekamy na zakończenie wszystkich wątków przy użyciu metody `.join()`. Blokuje ona główny wątek, aż wykonywanie odpowiedniego wątku się nie zakończy. Na końcu, zwracamy $4 \times$ `sum`, czyli pole, a dlatego że przyjęte jest, że $r = 1$, jest to π.

4.2. Implementacja kodu benchmarkującego

Funkcja ta też znajduje się w pliku `main.cpp`. Celem jest generowanie pliku `.csv` z wynikami benchmarków.

```
1  void generateBenchmarks(
2    const size_t minThreads,
3    const size_t maxThreads,
4    const size_t minIter,
5    const size_t maxIter
6  ) {
7    std::ofstream output;
8    output.open("benchmark.csv", std::ios::trunc | std::ios::out);
9
10   if(!output.is_open()) {
11     std::cerr << "Couldn't open benchmark file";
12     return;
13   }
```

```
14
15    output << "Threads;Iterations;Calculated;Time[ s]" << std::endl;
16
17    for(auto threads {minThreads}; threads <= maxThreads; threads++)
       {
18      size_t multiplier {10};
19
20      for(
21        size_t iterations {minIter};
22        iterations <= maxIter;
23        iterations >= 1'000'000'000 ? iterations += 1'000'000'000 :
      iterations *= multiplier
24      ) {
25        std::println("Benchmarking with {} threads and {} iterations.
      ", threads, iterations);
26
27        auto benchmarkStart { std::chrono::high_resolution_clock::now
      () };
28        auto calculated = getPi(iterations, threads);
29        auto benchmarkStop { std::chrono::high_resolution_clock::now
      () };
30
31        auto benchmarkDuration = std::chrono::duration_cast<std::
      chrono::microseconds>(
32          benchmarkStop - benchmarkStart
33        );
34
35        output << std::format("{};{};{};{}", threads, iterations,
      calculated, benchmarkDuration.count()) << std::endl;
36      }
37    }
38  }
```

Listing 3. Funkcja `generateBenchmarks()`

Jak widać na listingu nr. **??**, na linijkach nr. 1-6 funkcja przyjmuje parametry `minThreads`, określający minimalną liczbę wątków, `maxThreads`, określający maksymalną liczbę wątków, `minIter`, określający minimalną liczbę iteracji, `maxIter`, określający maksymalną liczbę iteracji. Na linijkach nr. 8-14 otwieramy plik `benchmark.csv` do zapisu. Jest to plik `.csv` ze strukturą `Threads;Iterations;Calculated;Time[µs]`. `Theads` określa liczbę wątków używaną podczas testu, `Iterations` liczbę iteracji, `Calculated` to wartość wykalkulowanej liczby π, a `Time[µs]` to czas pojedynczego pomiaru.

Na linijce nr. 17, zewnętrzna pętla `for` powtarza się `maxThreads - minThreads`

razy. To dlatego, że chcemy uzyskać pomiary dla wszystkich ilości wątków w danym przedziale. Zadeklarowana na linijce nr. 18 zmienna `multiplier` określa jak szybko należy zmieniać liczbę iteracji w funkcji `getPi()`. Mniejsze wartości `multiplier` pozwalają na większą dokładność, kosztem czasu wykonywania benchmarków. Na linijce nr. 20, wewnętrzna pętla `for` manipuluje ilością iteracji jaka ma być wykonywana. Maksymalnie może być to `maxIter` razy i minimalnie `minIter`, ale licznik `iterations` przy każdej iteracji jest mnożony razy 10 oraz jeżeli jest większy niż `1'000'000'000`, to dodawane jest `1'000'000'000`, ku celu oszczędzenia czasu. Następnie, przy użyciu biblioteki `std::chrono` mierzony jest czas przed i po wykonaniu funkcji oraz zapisywany jest czas trwania. Na linijce nr. 35 zapisywane są wyniki do pliku.

5. Wnioski

5.1. AI

Nie należy ignorować narzędzi AI jak Copilot i uznawać je jako jakieś tymczasowe zabawki dla deweloperów, które kiedyś wyjdą z mody. Nawet w swoim obecnym stanie, sam fakt, że po napisaniu `for`, jednym kliknięciem taba, *najprawdopodobniej* Copilot wytworzy nam sensowną pętle *biorąc pod uwagę kontekst szerszego programu* jest bez dwóch zdań bardzo użyteczny, po prostu przez to, że oszczędza to programiście czas. Ponadto, narzędzia te są świetnymi wyszukiwarkami, co szczególnie się tyczy modeli, które mają dostęp do internetu. Nie trzeba praktycznie walczyć z zareklamowanym po kark Google i przeszukiwać napisanych przez boty stron - taki model automatycznie "przesieje"internet i pokaże nam informacje, które faktycznie dotyczą naszego zapytania. No, chyba że o to co się pytamy jest rzeczą niszową.

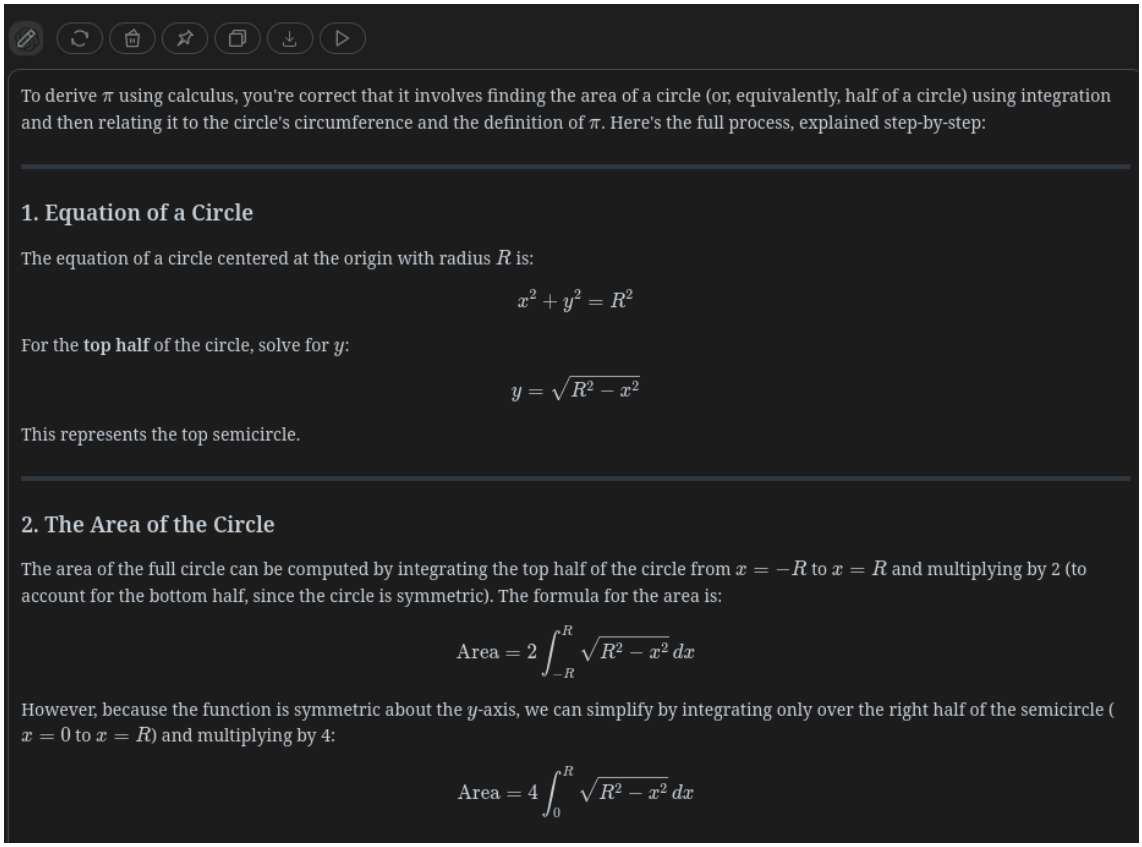
Niektórym (najbardziej to inwestorom w firmy zajmujące się AI) wydaje się, że LLM może za człowieka myśleć. Po części to prawda - jeżeli przed modelem o danym problemie myśleli inni - im więcej głowili się tym lepiej - i te przemyślenia gdzieś upublicznili, to LLM błyskawicznie może sięgnąć po tego problemu rozwiązanie. Ba, może nawet je po części zmodyfikować. Problem pojawia się, jeżeli zaczniemy naciskać na biednego chatbota. Fakty są następujące: LLM potrzebuje bardzo dużej ilości informacji o danym koncepcie, aby go opanować oraz LLM słabo potrafi rozumować, to jest, syntezować znane już informacje w nowe. Praktycznie, objawia się to gdy zapytamy się czatbota o rzecz, o której się mało mówi.

Osobiście mogę przytoczyć przykład próby zrozumienia API `pipewire`[5]. Jest to projekt zajmujący się zarządzaniem strumieniami audio (takimi z aplikacji) na Linuxie. Rzecz, z natury niszowa. Chciałem przechwycić wyjście jednej aplikacji i uzyskać na bieżąco sample jakie wysyła. Ile to było bawienia się z ChatemGPT, prób wyplucia przez niego programu który się nie segfaultuje - głównie dlatego, że nie chciało mi się czytać dokumentacji. Czat podczas treningu pewnie przeczytał - ale mało co z tego wyciągnął, bo strona z API to było jedno z niewielu miejsc, gdzie owo API było opisane. W końcu okazało się, że na stronie był przykład programu, który robił prawie to samo co chciałem i jedyne co ChatGPT mi dał to powód do przeczytania dokumentacji.

Pytać się można czy LLMy zastąpią programistów. Wydaje mi się, że - jeżeli nie teraz to za parę lat - programiści, których praca składa się z kopiowania zapytań o najnowszy framework JS ze Sack Overflowa i wklejania, faktycznie są zagrożeni, bo LLMy właściwie robią to, ale szybciej i taniej. Lecz programiści, którzy faktycznie

tworzą coś nowego, faktycznie tworzą projekty wcześniej niedokonane, stoją na czele innowacji lub nią są - nie mają się czym martwić na długo.

Dotknąłem już do czego funkcja czatu może być użyteczna. W tym akurat zadaniu użyłem ChataGPT do pokazania faktycznej matematyki za algorytmem. Widać jego wypowiedź na rysunku nr. **??**


To derive π using calculus, you're correct that it involves finding the area of a circle (or, equivalently, half of a circle) using integration and then relating it to the circle's circumference and the definition of π. Here's the full process, explained step-by-step:

1. Equation of a Circle

The equation of a circle centered at the origin with radius R is:

$$x^2 + y^2 = R^2$$

For the **top half** of the circle, solve for y:

$$y = \sqrt{R^2 - x^2}$$

This represents the top semicircle.

2. The Area of the Circle

The area of the full circle can be computed by integrating the top half of the circle from $x = -R$ to $x = R$ and multiplying by 2 (to account for the bottom half, since the circle is symmetric). The formula for the area is:

$$\text{Area} = 2 \int_{-R}^{R} \sqrt{R^2 - x^2}\, dx$$

However, because the function is symmetric about the y-axis, we can simplify by integrating only over the right half of the semicircle ($x = 0$ to $x = R$) and multiplying by 4:

$$\text{Area} = 4 \int_{0}^{R} \sqrt{R^2 - x^2}\, dx$$

Rys. 5.1. Wypowiedź Chata na temat sposobu obliczania pola pod kołem.

Wypowiedź z rysunku nr. **??** została wygenerowana modelem `gpt4o`.

5.2. Wykonanie programu

Program wykonuje się jak należy i kalkuluje π. Lecz warto zauważyć, że dla większej liczby wątków precyzja spada. Można to zauważyć na listingu nr. **??**

```
1 Select the accuracy of the algorithm (rectangles in a quarter of
     the circle): 10000000
2
3 Select the number of threads to use: 1
4
5 Calculated Pi: 3.141592853552536
6
```

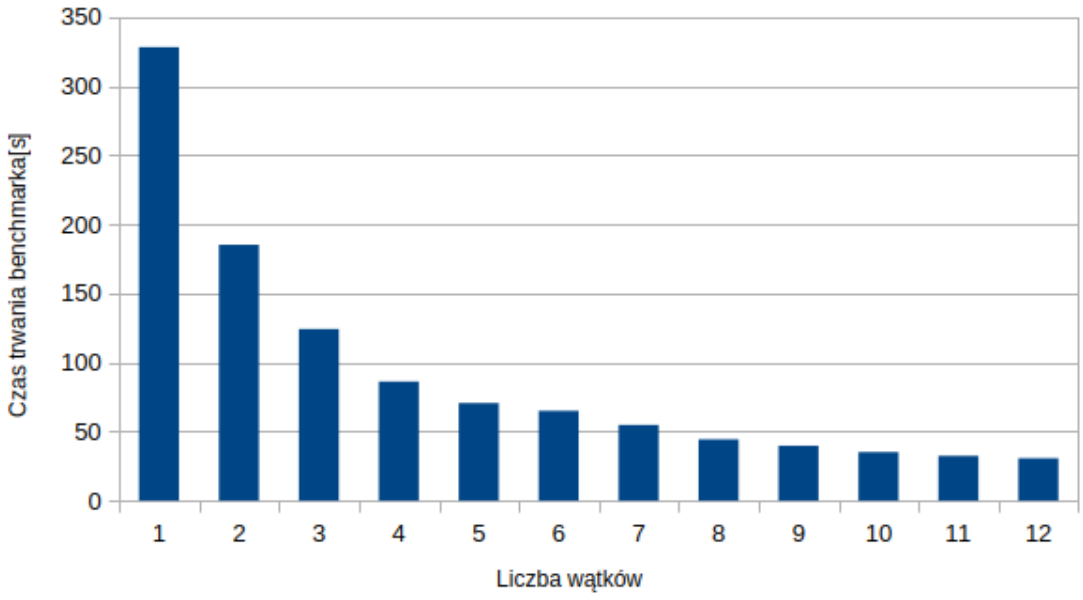
```
------

Select the accuracy of the algorithm (rectangles in a quarter of
    the circle): 10000000

Select the number of threads to use: 50

Calculated Pi: 3.1415936525118267
```

Listing 4. Funkcja `getPi()`

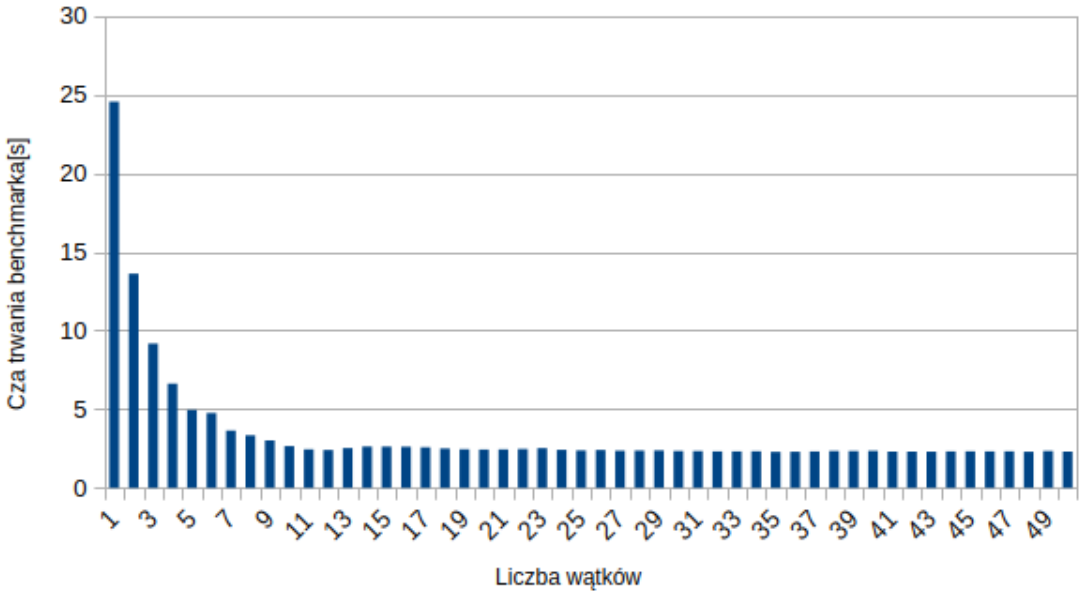
Wartość π to ok. 3.141592653589793. Porównując 13 i ostatnią linijkę, dla 50 wątków, wartość π jest mniej dokładna niż dla 1 wątku. Może to wynikać z tego, że wątki nie są synchronizowane, a więc mogą się zdarzyć przypadkowe kolizje, które zniekształcają wynik.

5.3. Benchmarki

Szersza implementacja benchmarków została opisana w sekcji nr. **??**. Benchmarki zostały przeprowadzone w dwóch zbiorach. Pierwszy, z wykresem na rysunku nr. **??** pokrył zakres od 1 do `5'000'000'000` iteracji i 1 do 12 wątków (tyle jest na maszynie testującej). Drugi wykres którego jest zawarty na rysunku nr. **??**, testował od 1 do 50 wątków, ale ograniczony był do `1'000'000'000` iteracji, w celu oszczędzenia czasu. Osi `y` wykresów to suma wszystkich iteracji funkcji benchmarkującej dla danej ilości wątków. Oś `x` to po prostu ilość wątków użyta w benchmarku.



Rys. 5.2. Benchmark z niepełnym zakresem wątków



Rys. 5.3. Benchmark z pełnym zakresem wątków

Jak widać na rysunku nr. **??** wraz ze zwiększeniem liczby wątków, czas benchmarku zmniejsza się wykładniczo. Na rysunku nr. **??** dzieje się podobnie, dopóki benchmark nie dojdzie do większej ilości wątków niż 12. Wtedy, czas wykonywania lekko wzrasta. Może się to dziać ponieważ tworzone jest więcej obiektów wątku, ale nie są one wykorzystywane przez procesor, co sprawia, że niepotrzebnie nakładany jest dodatkowy overhead.

Bibliografia

[1] *Artykul Wikipedii o macierzy*. URL: https://en.wikipedia.org/wiki/Matrix_(mathematics).

[2] *Strona Gita*. URL: https://git-scm.com/.

[3] *Strona Doxygena*. URL: https://www.doxygen.nl/.

[4] *Strona GitHub Copilot*. URL: https://github.com/features/copilot.

[5] *Strona PipeWire*. URL: https://pipewire.org/.

Spis rysunków

Spis tabel

Spis listingów