	**Politechnika Opolska** **Wydział Elektrotechniki, Automatyki i Informatyki** **Instytut Informatyki**
Rok akademicki	2023/2024
Przedmiot	Programowanie współbieżne i rozproszone
Forma zajęć	Laboratorium
Prowadzący zajęcia	mgr inż. Maciej Walczak
Grupa	**3**

Zadanie nr 1

Nazwisko i imię	Nr indeksu
Jakub Nieśmiała	**101374**

Uwagi

1. Charakterystyka zadania

Zadanie polega na stworzeniu programu w C++, który obliczy przybliżoną wartość całki oznaczonej funkcji kwadratowej w określonych granicach, wykorzystując trzy różne algorytmy całkowania. Do zrównoleglenia obliczeń należy użyć: OpenMP, C++ Thread Library oraz MPI. Wymagane jest przekazywanie współczynników funkcji oraz granic całkowania jako argumentów wiersza poleceń, skuteczne zrównoleglenie, brak błędów podczas uruchamiania i kończenia programu oraz napisanie raportu z realizacji zadania. Ponadto dodatkowe wymagania obejmują optymalną równomierną dystrybucję zadań w MPI, zdefiniowanie własnego typu danych MPI do przekazywania parametrów zadania oraz implementację trzech różnych algorytmów całkowania, które będzie można wybrać przez dodatkowy argument wiersza poleceń. W praktyce, program będzie pobierał dane wejściowe, przetwarzał je równolegle za pomocą odpowiednich bibliotek, a następnie zwracał wyniki. Implementacja musi uwzględniać zarówno aspekty równoległości, jak i poprawność numeryczną obliczeń całkowania. W raporcie znajdą się szczegółowe informacje o użytych rozwiązaniach oraz wynikach obliczeń.

$$I \approx \int_a^b (Ax^2 + Bx + C)dx$$

2. Opis środowiska

1. **Procesor**: Intel Core i5-11400H
    - Model procesora: Intel Core i5-11400H
    - Liczba rdzeni fizycznych: 6
    - Liczba procesorów logicznych(wątków): 12
    - Częstotliwość bazowa: 2.7 GHz
2. **Pamięć RAM**: 16 GB DDR4
    - Pojemność: 16 GB
    - Typ: DDR4
    - Prędkość: Standardowa szybkość dla pamięci DDR4
3. **System operacyjny**: Windows 11
    - Wersja systemu operacyjnego: Windows 11 Home 23H2

- Architektura systemu: x64 (64-bitowa)

4. **Oprogramowanie i narzędzia programistyczne**:
    - Środowisko programistyczne: Visual Studio
    - Wersja Visual Studio: 2022
    - Język programowania: C++
    - Biblioteki i narzędzia:
        - MPI 4.1.1
        - OpenMP: 5.0.
        - C++ Threads: C++17

3. Opis zastosowanych technologii oraz algorytmu

W każdym z trzech wariantów kodu zastosowany został numeryczny sposób całkowania funkcji kwadratowej, którego celem jest przybliżone obliczenie wartości całki oznaczonej na zadanym przedziale. Wybrane do implementacji zostały następujące metody obliczania całki oznaczonej:

1. Metoda trapezów (trapezoidal): Podobnie jak metoda prostokątów, ale zamiast prostokątów używa trapezów do przybliżenia pola pod krzywą.
2. Metoda środka tzw. metoda prostokątów (midpoint): Dzieli przedział całkowania na n równych części i przybliża pole pod krzywą jako sumę pól prostokątów.
3. Metoda Simpsona (simpson): Dzieli przedział całkowania na n równych części i oblicza pole pod krzywą, korzystając z przybliżonego wzoru Simpsona.

Sposób dekompozycji i dystrybucji zadania oraz zasoby współdzielone przedstawione są poniżej dla każdego wariantu zrównoleglania kodu:

1. **MPI**
    - Dekompozycja i dystrybucja: Proces o rankingu 0, czyli master, odpowiada za pobranie wartości z wiersza poleceń. Te wartości reprezentują parametry zadania. Proces master konwertuje te wartości na odpowiedni typ danych i przechowuje je w specjalnie stworzonej do tego strukturze oraz rozgłasza parametry (własną strukturę danych) do wszystkich procesów. Następnie algorytm wykonuje się równolegle przez wiele procesów MPI, gdzie każdy oblicza część całki dla swojego podprzedziału. Na koniec proces master, po upewnieniu się, że wszystkie procesy zakończyły swoje obliczenia, sumuje ich wyniki.

- Zasoby współdzielone: Współdzielone są dane wejściowe (parametry zadania), które są rozgłaszane na wszystkie procesy w specjalnej strukturze.

2. **OpenMP**
   - Dekompozycja i dystrybucja: W każdej z metod całkowania (trapezoidalnej, środkowej i Simpsona) pętla obliczająca wyniki dla podprzedziałów jest równolegle dzielona między wątkami. Każdy wątek oblicza więc cześć sumy lokalnej, a wyniki są sumowane do wspólnej zmiennej sumy.
   - Zasoby współdzielone: Współdzielona w każdej metodzie jest zmienna sumy, pozwala to na wykorzystanie mechanizmu redukcji, co zapewnia bezpieczną operację dodawania wyników częściowych.

3. **C++ thread Library**
   - Dekompozycja i dystrybucja: Wątki są tworzone w zależności od liczby wątków podanych przez użytkownika. Każdy wątek oblicza wartości dla swoich podprzedziałów oraz sumuje je w sumie lokalnej, a wyniki każdego wątku są dodawane do wspólnej zmiennej sumy za pomocą obiektu mutex.
   - Zasoby współdzielone: Współdzielona jest jedynie zmienna sumy, która jest chroniona przed równoczesnym dostępem za pomocą mechanizmu obiektu mutex.

Ponadto w implementacji zastosowane zostały różne mechanizmy MPI, OpenMP oraz C++ Thread Library umożliwiające zrównoleglenie obliczeń. Opisy wykorzystanych mechanizmów dla każdego wariantu programu przedstawione są poniżej:

1. **MPI**
   - MPI_Init - Mechanizm inicjujący MPI, który przygotowuje środowisko MPI do pracy,
   - MPI_Comm_rank - Pobiera numer identyfikacyjny (rank) bieżącego procesu wewnątrz komunikatora MPI_COMM_WORLD,
   - MPI_Comm_size - Pobiera liczbę procesów biorących udział w komunikacji wewnątrz komunikatora MPI_COMM_WORLD,
   - MPI_Bcast - Mechanizm rozgłaszania, który przesyła dane z jednego procesu (w tym przypadku procesu o ranku 0) do wszystkich innych procesów,
   - MPI_Barrier - Mechanizm synchronizacji, który wymusza oczekiwanie na wszystkich procesach do momentu, gdy każdy z nich osiągnie daną linijkę kodu,
   - MPI_Reduce - Mechanizm redukcji, który zbiera dane z różnych procesów i wykonuje operację na tych danych (np. sumę),
   - MPI_Finalize() - Mechanizm kończący pracę z MPI, czyszczący i zamykający wszystkie zasoby używane przez MPI.

2. **OpenMP**
   - #pragma omp parallel for - Ta dyrektywa tworzy region równoległy, gdzie każdy wątek wykonuje pętlę for równolegle,

- reduction(+:sum) - Ta klauzula pozwala na to, że każdy wątek ma swoją kopię zmiennej sum, a po zakończeniu regionu równoległego, wartości tych zmiennych są zredukowane (tutaj zsumowane) do jednej wartości. Dzięki temu unika się konfliktów w dostępie do zmiennej współdzielonej,
- omp_get_wtime() - Służy do pobrania czasu wykonywania się programu.

3. **C++ thread Library**
   - thread - Jest to klasa reprezentująca pojedynczy wątek wykonawczy. W kodzie użyto klasy std::thread do tworzenia i zarządzania wieloma wątkami, które są odpowiedzialne za równoległe obliczenia numeryczne,
   - mutex - jest klasą reprezentującą mutex (muteks), który służy do synchronizacji dostępu do współdzielonych zasobów przez wiele wątków. W kodzie użyto mutexów, aby zabezpieczyć dostęp do współdzielonych zmiennych, takich jak suma całkowania, zapobiegając w ten sposób równoczesnemu zapisowi przez wiele wątków.

4. Opis scenariuszy testowych i metodologii badań

Dla każdej metody całkowania (metoda środka(tzw. prostokątów), metoda trapezów, metoda Simpsona) przeprowadzono testy jednostkowe, aby zweryfikować poprawność obliczeń. Wybrano kilka przykładowych funkcji kwadratowych o różnych współczynnikach oraz przedziałach całkowania. Wyniki uzyskane przez każdą metodę porównano z oczekiwanymi wartościami całek, obliczonymi przy użyciu zaufanego narzędzia do obliczeń całkowych.

Dla każdej metody całkowania przeprowadzono testy wydajnościowe z różnymi rozmiarami danych wejściowych (liczba podziałów przedziału całkowania). Wybrano kilka funkcji kwadratowych oraz przedziałów całkowania o różnych rozmiarach. Dokonano pomiaru czasu wykonania dla każdej metody oraz obliczono średnie czasy wykonania wszystkich metod całkowania dla każdej wersji programu przy użyciu różnych rozmiarów danych wejściowych. Następnie porównano czasy wykonania dla różnych wersji programu w zależności od rozmiaru danych wejściowych.

Metodologia badań:

1. **Przygotowanie środowiska:**

   - Należy skonfigurować środowisko testowe dla każdej metody, zapewniając odpowiednie kompilatory (np. GCC dla OpenMP i C++ Thread Library, MPI dla MPI), biblioteki i zasoby sprzętowe.

   - Należy upewnij się, że wszelkie zależności i środowiska uruchomieniowe są poprawnie skonfigurowane i dostępne.

2. **Przeprowadzenie testów:**

- Przeprowadź testy jednostkowe dla każdej metody całkowania, sprawdzając poprawność obliczeń dla różnych funkcji kwadratowych i przedziałów całkowania.
- Następnie przeprowadź testy wydajnościowe, mierząc czas wykonania dla każdej metody przy różnych rozmiarach danych wejściowych.
- Upewnij się, że wszystkie testy są przeprowadzane na tym samym sprzęcie i w tych samych warunkach środowiskowych, aby wyniki były porównywalne.

3. **Analiza wyników:**
   - Należy dokładnie przeanalizować wyniki testów, sprawdzając poprawność obliczeń oraz porównując czasy wykonania dla różnych metod i rozmiarów danych wejściowych.
   - Na podstawie testów należy określić, która wersja zrównoleglenia programu najlepiej radzi sobie z różnymi rodzajami funkcji i rozmiarami danych.
   - Dokonaj wniosków dotyczących wydajności i skuteczności każdej wersji programu.

5. Wyniki badań

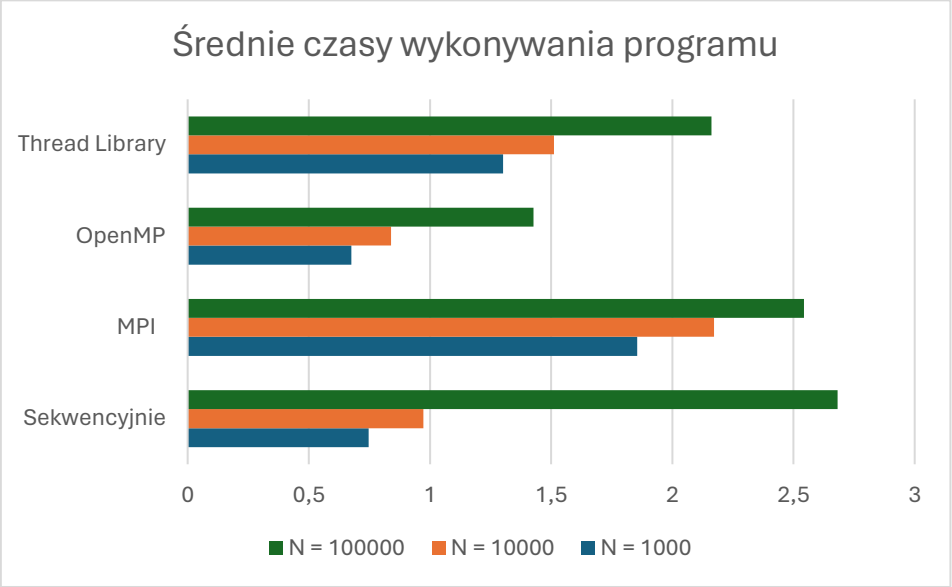
Tabela z średnimi czasami wykonania dla różnych wersji programu i wartości parametru n:

Metoda	**N = 1000**	**N = 10000**	**N = 100000**
Sekwencyjnie	0,7468 ms	0,9733 ms	2,6822 ms
MPI	1,8546 ms	2,1726 ms	2,5436 ms
OpenMP	0,5748 ms	0,8394 ms	1,4275 ms
Thread Library	1,302 ms	1,512 ms	2,161 ms

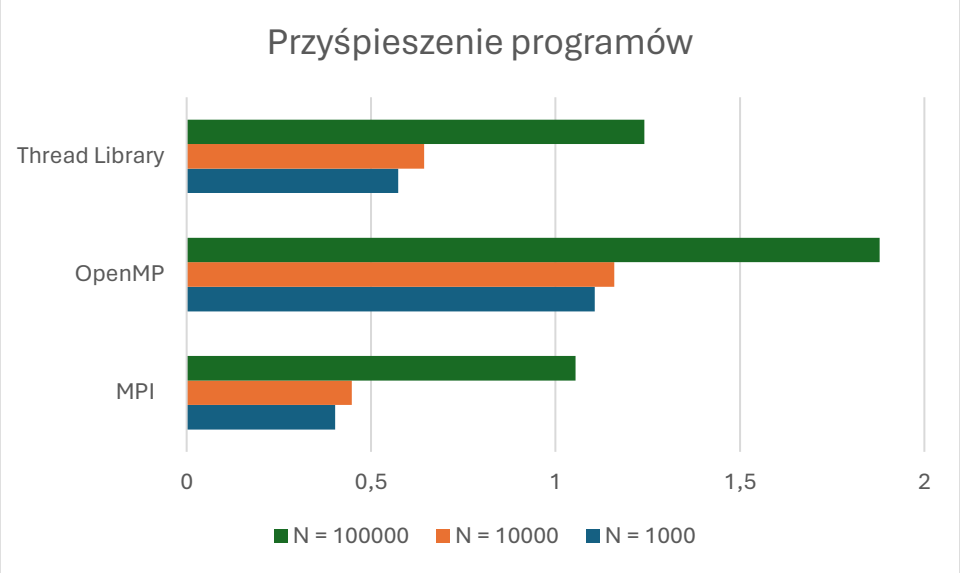
Tabela z przyśpieszeniem dla różnych wersji programu i wartości parametru n w stosunku do metody sekwencyjnej:

Metoda	**N = 1000**	**N = 1000**	**N = 100000**
MPI	0,402674x	0,447989x	1,05449x
OpenMP	1,299235x	1,159519x	1,878949x
Thread Library	0,573579x	0,643717x	1,241185x

Wykres przedstawiający średnie czasy wykonania dla różnych wersji programu i wartości parametru n:



Wykres przedstawiający przyśpieszeniem dla różnych wersji programu i wartości parametru n w stosunku do metody sekwencyjnej:



Oszacowanie przyśpieszenia:

- Na podstawie danych empirycznych możemy obliczyć przyśpieszenie jako stosunek czasu wykonania algorytmu sekwencyjnego do czasu wykonania algorytmu równoległego. Przykładowo, dla średniej wartości z metod dla wersji MPI i n = 100000 podziałów: Przyśpieszenie = Czas_sekwencyjny / Czas_równoległy = 2,6822 ms / 2,5436 ms = 1,05449x
    - MPI: 1,05449x
    - OpenMP: 1,87895x

    - Thread Library: 1,24118

- Prawo Amdahla pozwala oszacować maksymalne teoretyczne przyśpieszenie algorytmu równoległego. Przyśpieszenie można obliczyć zgodnie ze wzorem: Przyśpieszenie = 1 / ((1 - P) + (P / S)), gdzie P to frakcja kodu, która może być wykonana równolegle, a S to liczba procesorów. Jeśli założymy, że np. 92% kodu może być wykonane równolegle (P = 0.92) i korzystamy z 12 rdzeni CPU (S = 12), to oszacowane przyśpieszenie wynosi: Przyśpieszenie = 1 / ((1 - 0.9) + (0.9 / 12)) ≈ 5.7142x

Próg opłacalności realizacji równoległej:

Próg opłacalności implementacji równoległej można określić jako moment, kiedy przyspieszenie algorytmu równoległego przewyższa czas wykonania algorytmu sekwencyjnego dla określonego zestawu danych wejściowych. Można te wartości oszacować poprzez zastosowanie interpolacji liniowej dwóch wielkości zadania dla których przyśpieszenie jest możliwie bliskie wartości 1.0x:

- MPI $n_{min} \approx 91800$
- OpenMP $n_{min} \approx 768$
- C++ Thread Library $n_{min} \approx 870$

6. Wnioski

Przeprowadzone badania skupiły się na efektywności różnych metod liczenia całek oznaczonych w zrównoleglonych wersjach programu. Wszystkie zaimplementowane metody zapewniają poprawne przybliżenie wartości dla określonych argumentów. Równoległe wersje programu zostały pomyślnie opracowane, a dla dużych wielkości zadania każda z nich działa szybciej niż odpowiednik sekwencyjny. Każdy z programów działa bezbłędnie i umożliwia pomiar czasu realizacji dla określonych argumentów zadania. Analiza wyników wykazała, że najbardziej efektywną metodą jest implementacja równoległa algorytmu przy użyciu OpenMP, natomiast implementacja z biblioteką Threads C++ zajmuje drugie miejsce pod względem szybkości. Z kolei implementacja z użyciem MPI uzyskała najmniejszą wydajność w porównaniu z pozostałymi metodami, MPI pozwala jednak na przeprowadzanie obliczeń na wielu komputerach (procesorach) poprzez tworzenie klastrów i komunikowanie się pomiędzy nimi co daje spory potencjał mimo słabego wyniku w tym badaniu.