


| | |
|---|--|
|  | 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 3

| Nazwisko i imię | Nr indeksu |
|-----------------|------------|
| Jakub Nieśmiata | 101374 |

Uwagi

1. Charakterystyka zadania

Zadanie polega na stworzeniu programu w C++, który współbieżnie rozwiązuje układ równań liniowych używając technologii: OpenMP oraz biblioteki Threads C++. Program pobiera rozszerzoną macierz współczynników z pliku CSV, rozwiązuje układ równań metodą Gaussa lub Gaussa-Jordana wraz z wyborem elementu podstawowego, a następnie zapisuje wektor wynikowy do nowego pliku CSV. Dodatkowo, program musi obsługiwać parametry przekazywane przez wiersz poleceń oraz umożliwiać kontrolę liczby wątków lub procesów używanych przez program.

Metoda eliminacji Gaussa

Metoda eliminacji Gaussa jest algorytmem używanym do rozwiązywania układów równań liniowych. Polega na przekształceniu macierzy układu do postaci trójkątnej górnej poprzez eliminację elementów poniżej głównego elementu (pivotu) w każdej kolumnie. Następnie rozwiązanie układu uzyskuje się poprzez podstawianie wsteczne, zaczynając od ostatniego równania i stopniowo obliczając wartości zmiennych.

Metoda eliminacji Gaussa-Jordana

Metoda eliminacji Gaussa-Jordana rozszerza metodę Gaussa, przekształcając macierz układu równań do postaci diagonalnej. W tej metodzie nie tylko eliminujemy elementy poniżej pivotu, ale również te powyżej, przekształcając każdy pivot do jedynki. Dzięki temu bezpośrednio otrzymujemy rozwiązanie układu równań z macierzy w postaci jednostkowej, gdzie jedynki są na przekątnej, a reszta elementów to zera.

Pivoting częściowy

Pivotowanie częściowe jest techniką stosowaną podczas eliminacji Gaussa oraz eliminacji Gaussa-Jordana w celu poprawy stabilności numerycznej. Polega na wyborze największego elementu w bieżącej kolumnie, począwszy od wiersza pivotu w dół, a następnie zamianie wierszy, aby ten największy element znalazł się na przekątnej. Dzięki temu zmniejsza się ryzyko dzielenia przez bardzo małe liczby, co poprawia dokładność obliczeń.

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

Algorytm mnożenia macierzy w programie zawiera kilka kluczowych etapów:

1. **Wczytanie danych wejściowych:** Program wczytuje macierz współczynników z pliku CSV za pomocą funkcji `readMatrixFromFile()`. Macierz ta zawiera współczynniki równań oraz wyrazy wolne.

2. **Inicjalizacja parametrów:** Program pobiera parametry wejściowe, takie jak liczba wątków, metoda rozwiązywania (G dla Gaussa lub GJ dla Gaussa-Jordana), oraz nazwa pliku logu.
3. **Wybór metody rozwiązywania:** Na podstawie przekazanego parametru *method*, program wybiera odpowiednią funkcję do przekształcania macierzy:
 - *gaussElimination()* dla metody eliminacji Gaussa.
 - *gaussJordanElimination()* dla metody Gaussa-Jordana.
4. **Eliminacja Gaussa:**
 - **Pivotowanie częściowe(opcjonalne):** W każdym kroku eliminacji wybierany jest największy element w bieżącej kolumnie poniżej głównego elementu (pivotu) i zamieniany miejscami z pivotem.
 - **Eliminacja w przód:** Przekształcenie macierzy do postaci trójkątnej górnej przez eliminację elementów poniżej pivotu.
 - **Wsteczna substytucja:** Obliczanie wartości zmiennych, zaczynając od ostatniego równania i przesuwając się wstecz.
5. **Eliminacja Gaussa-Jordana:**
 - **Pivotowanie częściowe(opcjonalne):** Podobnie jak w metodzie Gaussa, wybierany jest największy element w bieżącej kolumnie poniżej pivotu i zamieniany miejscami z pivotem.
 - **Eliminacja w przód i wstecz:** Eliminacja elementów zarówno poniżej, jak i powyżej pivotu, przekształcając macierz do postaci jednostkowej.
6. **Logowanie i zapisywanie wyników:** Program zapisuje rozwiązany wektor do pliku CSV za pomocą funkcji *writeVectorToFile()*. Szczegóły wykonania, takie jak liczba równań, liczba wątków, metoda oraz czas wykonania, są logowane do pliku logu przy użyciu funkcji *logExecutionDetails()*.
7. **Pomiar czasu wykonania:** Czas wykonania jest mierzony od momentu rozpoczęcia eliminacji do jej zakończenia, a następnie konwertowany na milisekundy i logowany.

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

1. OpenMP

- Rozwiązywanie układu równań jest realizowane równolegle przy użyciu dyrektyw OpenMP. Dekompozycja zadania polega na podziale pętli eliminacji i substytucji wstecznej na różne wątki. Segment kodu odpowiedzialny za pivotowanie oraz

normalizację wiersza pivotu jest wykonywany tylko przez jeden wątek(pierwszy który dojdzie do tego fragmentu kodu) dzięki dyrektywie `#pragma omp single`. Zasobami współdzielonymi oprócz zmiennych reprezentujących ilość wątków i informacje czy funkcja ma pivotować są macierz wejściowa oraz wektor wyników do których wątki zapisują wyniki z poszczególnych układów równań(przy wstecznej substytucji każdy wątek ma swój zakres równań do rozwiązania i swój zakres wektora w którym zapisuje wyniki)

2. C++ thread Library

- Wątki są tworzone w zależności od liczby wątków podanych przez użytkownika. Dekompozycja zadania polega na podziale pętli eliminacji i substytucji wstecznej na różne wątki. Segment kodu odpowiedzialny za pivotowanie oraz normalizację wiersza pivotu jest zamknięty w sekcji krytycznej (mutex i lock_guard) aby uniknąć konfliktów przy jednoczesnym dostępie wielu wątków do wspólnych zasobów. Zasobami współdzielonymi oprócz zmiennych reprezentujących ilość wątków i informacje czy funkcja ma pivotować są macierz wejściowa oraz wektor wyników do których wątki zapisują wyniki z poszczególnych układów równań(przy wstecznej substytucji każdy wątek ma swój zakres równań do rozwiązania i swój zakres wektora w którym zapisuje wyniki)

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

1. OpenMP

- `#pragma omp parallel for` - Ta dyrektywa tworzy region równoległy, gdzie każdy wątek wykonuje pętlę `for` równoległe,
- `collapse(collapseLevel)` – dyrektywa ta pozwala na zagnieżdżanie wielu pętli w jednej równoległej sekcji, zwiększając poziom równoległości.
- `schedule(scheduleType)` - Różne harmonogramy, takie jak `static`, `dynamic` i `guided`, kontrolują sposób przydziału iteracji pętli do wątków.
- `omp atomic` – dyrektywa ta zapewnia atomowe wykonanie operacji na zmiennych w celu uniknięcia problemów z dostępem wielu wątków do tych samych danych.
- `omp_get_wtime()` - Służy do pobrania czasu wykonywania się programu.

2. 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.

- `lock_guard` - Użycie `lock_guard` do automatycznego zarządzania blokowaniem i odblokowywaniem mutexa w ramach zakresu, zapewniając jednocześnie wykonanie tylko jednego wątku w sekcji krytycznej.

3. Opis scenariuszy testowych i metodologii badań

Scenariusze testowe:

1. Poprawność wyników:

- Sprawdzenie zgodności wyników z obliczeń sekwencyjnych dla małych danych.
- Porównanie wyników dla różnych przypadków testowych.

2. Wydajność:

- Pomiar czasu wykonania dla różnych liczb wątków lub procesów.
- Porównanie czasów wykonania dla różnych implementacji.

3. Skalowalność:

- Badanie czasu wykonania dla rosnącej liczby wątków lub procesów.
- Analiza wydajności w zależności od zmiennej liczby wątków lub procesów.

Metodologia badań:

1. Testowanie poprawności wyników:

- Porównanie wyników z obliczeniami sekwencyjnymi dla małych danych.
- Analiza wyników dla różnych przypadków testowych.

2. Testowanie wydajności:

- Pomiar czasu wykonania dla różnych konfiguracji wątków lub procesów.
- Porównanie czasów wykonania między implementacjami.

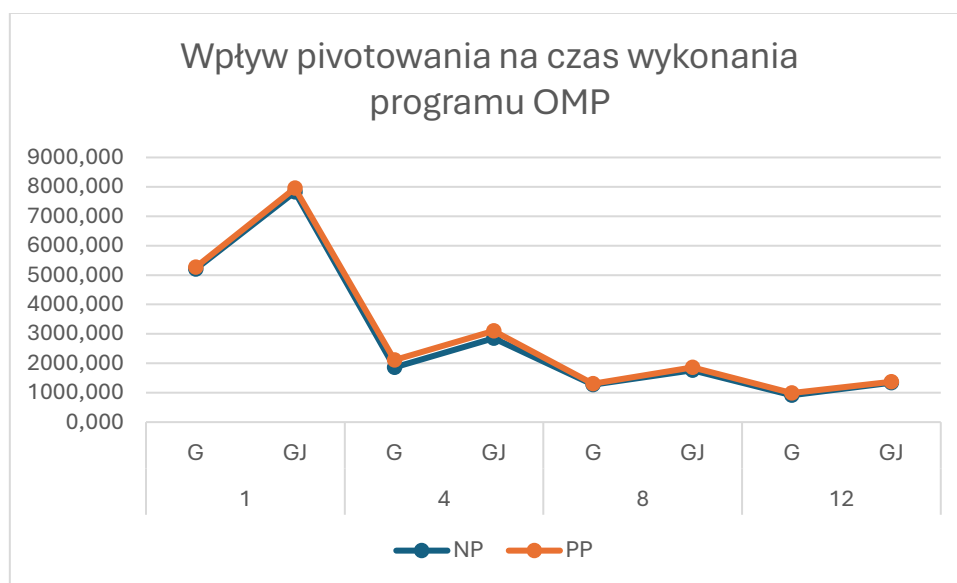
3. Testowanie skalowalności:

- Badanie czasu wykonania dla rosnącej liczby zasobów.
- Analiza wydajności w zależności od liczby wątków lub procesów.

4. Wyniki badań

Wpływ zastosowania algorytmu częściowego pivotowania dla implementacji OpenMP na największym zbiorze danych dla obu metod rozwiązywania układów równań:

| Wpływ pivotowania na czas wykonania implementacji OMP | | | | | | | | |
|---|----------|----------|----------|----------|----------|----------|---------|----------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| NP | 5199,335 | 7824,966 | 1859,962 | 2856,054 | 1276,578 | 1762,298 | 913,217 | 1333,169 |
| PP | 5268,071 | 7953,141 | 2107,953 | 3103,831 | 1308,139 | 1864,25 | 988,46 | 1374,24 |



Dla zastosowanych danych testowych program nie wykazał różnic w wartościach (do 6 znaków po przecinku) rozwiązując układ z pivotowaniem lub bez, dlatego tabela z różnicami oraz średnimi różnicami absolutnymi zostały pominięte.

Wszystkie pozostałe testy dla obu metod wykonane zostały z zastosowaniem pivotowania częściowego. Dla każdej implementacji programu zostało wykonane 10 testów dla każdej wielkości zadania i każdej metody, a następnie został obliczony średni czas wykonywania danego przypadku przedstawione poniżej w tabelkach.

Tabela z średnimi czasami wykonania dla OpenMP wyrażonymi w milisekundach dla obu metod:

| Ilość równań w układzie | Liczba wątków | | | | | | | |
|-------------------------|---------------|----------|----------|----------|----------|----------|---------|----------|
| | 1 | | 4 | | 8 | | 12 | |
| | G | GJ | G | GJ | G | GJ | G | GJ |
| 50 | 0,739 | 1,116 | 1,067 | 1,126 | 1,250 | 1,312 | 1,367 | 1,608 |
| 250 | 96,013 | 118,206 | 37,437 | 55,312 | 28,093 | 32,756 | 22,487 | 26,652 |
| 500 | 645,009 | 987,049 | 234,590 | 401,186 | 155,725 | 225,777 | 120,628 | 167,562 |
| 1000 | 5268,071 | 7953,141 | 2107,953 | 3103,831 | 1308,139 | 1864,250 | 988,460 | 1374,240 |

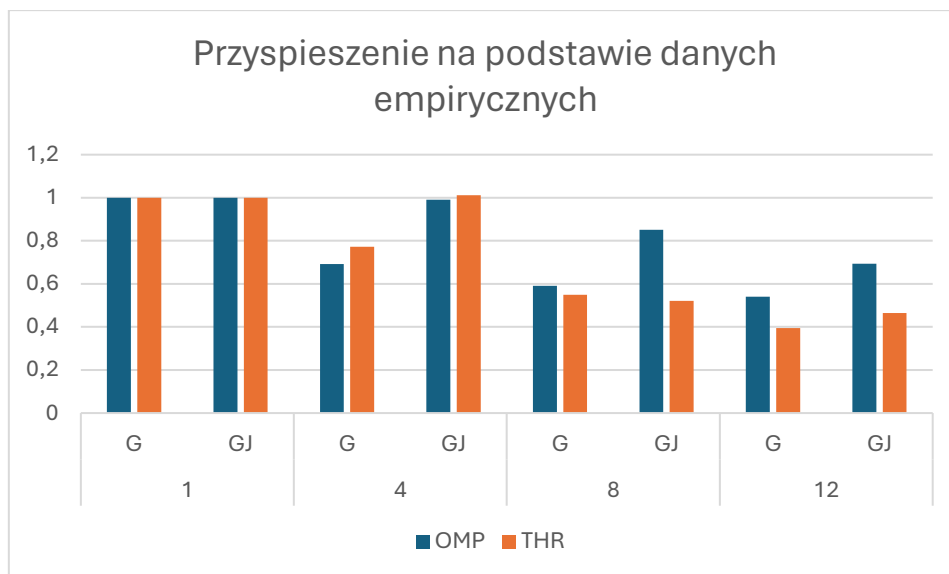
Tabela z średnimi czasami wykonania dla klasy Threads wyrażonymi w milisekundach dla obu metod:

| Ilość równań w układzie | Liczba wątków | | | | | | | |
|-------------------------|---------------|----------|----------|----------|----------|----------|----------|----------|
| | 1 | | 4 | | 8 | | 12 | |
| | G | GJ | G | GJ | G | GJ | G | GJ |
| 50 | 14,297 | 17,533 | 18,513 | 17,335 | 26,030 | 33,622 | 36,251 | 37,793 |
| 250 | 148,270 | 193,082 | 104,074 | 119,242 | 119,308 | 129,009 | 158,354 | 176,278 |
| 500 | 741,651 | 1109,739 | 442,869 | 584,213 | 410,452 | 468,477 | 378,494 | 449,050 |
| 1000 | 5665,511 | 7769,227 | 2450,111 | 3510,351 | 1732,705 | 2580,083 | 1575,903 | 2015,759 |

Oszacowanie przyspieszenia na podstawie danych empirycznych:

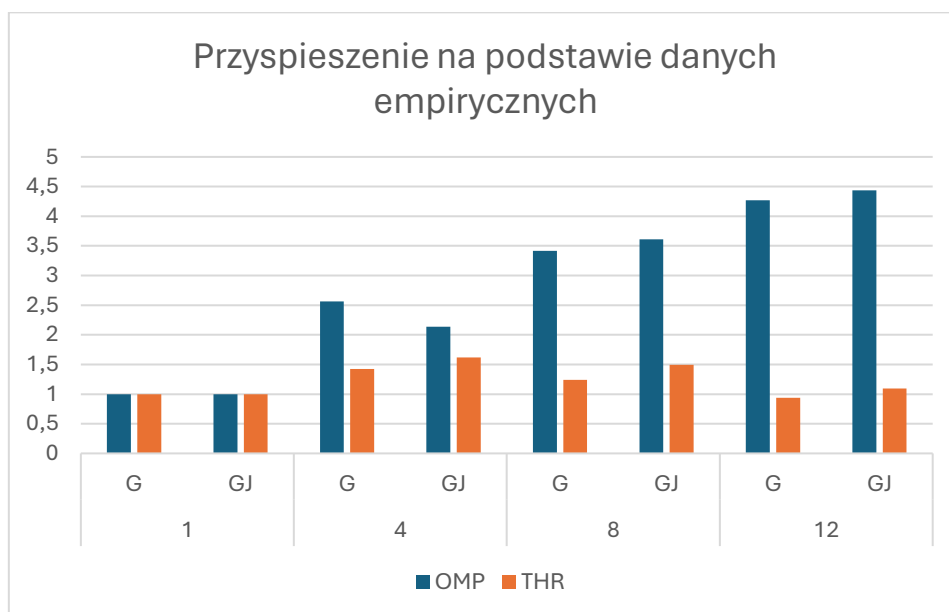
- Dla układu 50 równań dla obu metod:

| Przyspieszenie na podstawie danych empirycznych [ms] | | | | | | | | |
|--|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | 1 | 1 | 0,693 | 0,991 | 0,591 | 0,851 | 0,541 | 0,694 |
| THR | 1 | 1 | 0,772 | 1,011 | 0,549 | 0,521 | 0,394 | 0,464 |



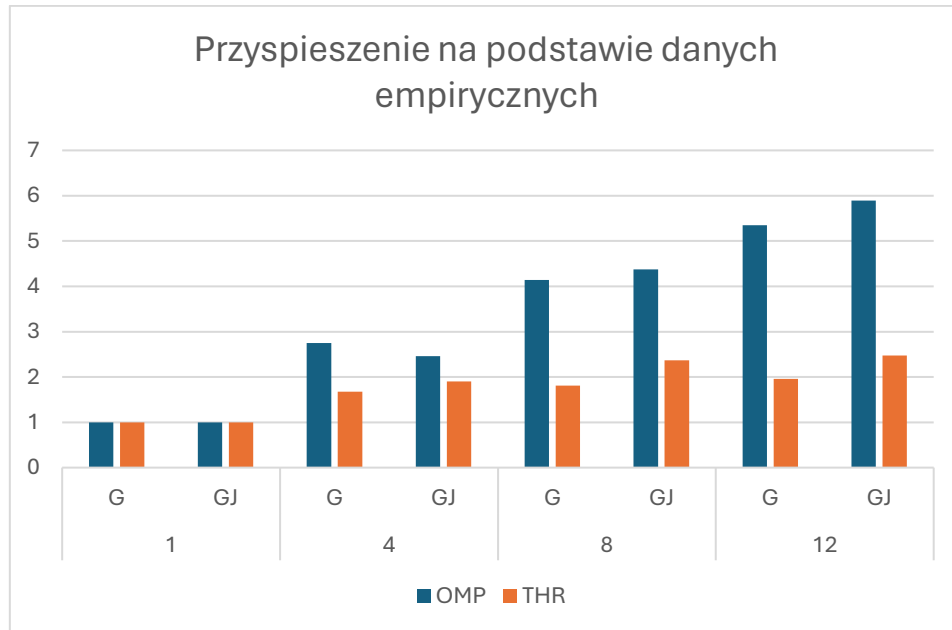
- Dla układu 250 równań dla obu metod:

| Przyspieszenie na podstawie danych empirycznych [ms] | | | | | | | | |
|--|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | 1 | 1 | 2,565 | 2,137 | 3,418 | 3,609 | 4,270 | 4,435 |
| THR | 1 | 1 | 1,425 | 1,619 | 1,243 | 1,497 | 0,936 | 1,095 |



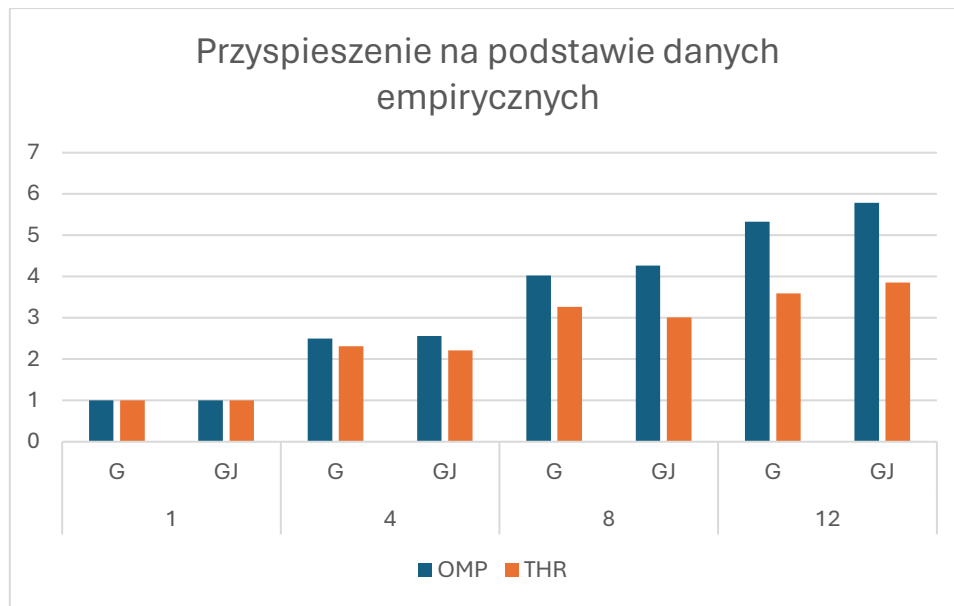
- Dla układu 500 równań dla obu metod:

| Przyspieszenie na podstawie danych empirycznych [ms] | | | | | | | | |
|--|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | 1 | 1 | 2,750 | 2,460 | 4,142 | 4,372 | 5,347 | 5,891 |
| THR | 1 | 1 | 1,675 | 1,900 | 1,807 | 2,369 | 1,959 | 2,471 |



- Dla układu 1000 równań dla obu metod:

| Przyspieszenie na podstawie danych empirycznych [ms] | | | | | | | | |
|--|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | 1 | 1 | 2,499 | 2,562 | 4,027 | 4,266 | 5,330 | 5,787 |
| THR | 1 | 1 | 2,312 | 2,213 | 3,270 | 3,011 | 3,595 | 3,854 |



Prawo Amdahla wymaga określenia stopnia zrównoleglenia danego zadania oraz uwzględnienia części sekwencyjnej.

Przyspieszenie można wyrazić wzorem:

$$S(p) = \frac{1}{(1-P) + \frac{P}{p}}$$

gdzie:

- S(p) to przyspieszenie,
- P to stopień zrównoleglenia,
- p to liczba rdzeni lub wątków.

P = (Czas Sekwencyjny - Czas Równoległy) / Czas Sekwencyjny

Prawo Amdahla = 1 / ((1-P) + (P/Liczba wątków))

Przyspieszenie z prawa Amdahla:

- Dla układu 50 równań dla obu metod:

| Przyspieszenie na podstawie prawa Amdahla [ms] | | | | | | | | |
|--|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | 1 | 1 | 0,750 | 0,993 | 0,623 | 0,867 | 0,562 | 0,712 |
| THR | 1 | 1 | 0,819 | 1,009 | 0,582 | 0,555 | 0,415 | 0,486 |

- Dla układu 250 równań dla obu metod:

| Przyspieszenie na podstawie prawa Amdahla [ms] | | | | | | | | |
|--|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | 1 | 1 | 1,844 | 1,664 | 2,625 | 2,721 | 3,355 | 3,448 |
| THR | 1 | 1 | 1,288 | 1,402 | 1,206 | 1,409 | 0,941 | 1,087 |

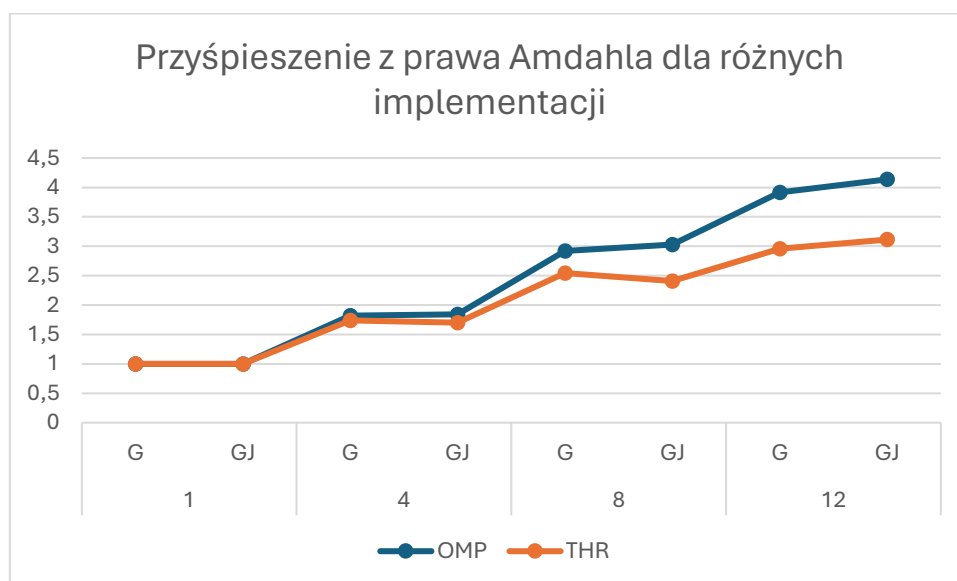
- Dla układu 500 równań dla obu metod:

| Przyspieszenie na podstawie prawa Amdahla [ms] | | | | | | | | |
|--|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | 1 | 1 | 1,913 | 1,802 | 2,974 | 3,076 | 3,925 | 4,185 |
| THR | 1 | 1 | 1,433 | 1,551 | 1,641 | 2,023 | 1,814 | 2,201 |

- Dla układu 1000 równań dla obu metod:

| Przyspieszenie na podstawie prawa Amdahla [ms] | | | | | | | | |
|--|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | 1 | 1 | 1,818 | 1,843 | 2,922 | 3,029 | 3,917 | 4,137 |
| THR | 1 | 1 | 1,741 | 1,698 | 2,547 | 2,406 | 2,956 | 3,114 |

Wartości dla prawa Amdahla uzyskane dla największego zestawu danych dla obu metod przedstawione na wykresie:



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.

Próg opłacalności = czas wykonania sekwencyjnego algorytmu / (Czas wykonania sekwencyjnego - Czas wykonania równoległego)

- Dla układu 50 równań dla obu metod:

| Progi opłacalności implementacji równoległej [ms] | | | | | | | | |
|---|---|----|--------|----------|--------|--------|--------|--------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | - | - | -2,253 | -111,600 | -1,446 | -5,694 | -1,177 | -2,268 |
| THR | - | - | -3,391 | 88,551 | -1,219 | -1,090 | -0,651 | -0,865 |

- Dla układu 250 równań dla obu metod:

| Progi opłacalności implementacji równoległej [ms] | | | | | | | | |
|---|---|----|-------|-------|-------|-------|---------|--------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | - | - | 1,639 | 1,879 | 1,414 | 1,383 | 1,306 | 1,291 |
| THR | - | - | 3,355 | 2,615 | 5,119 | 3,013 | -14,703 | 11,490 |

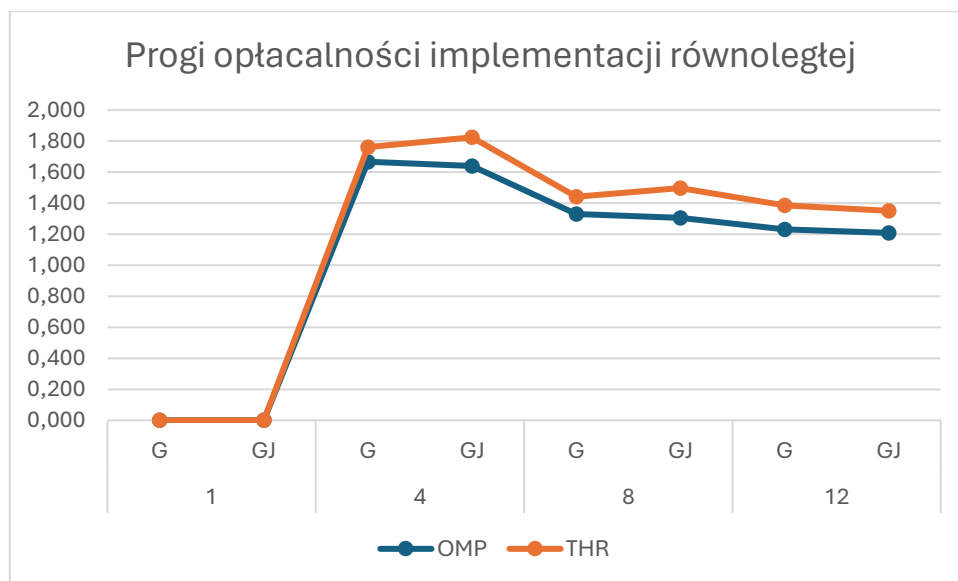
- Dla układu 500 równań dla obu metod:

| Progi opłacalności implementacji równoległej [ms] | | | | | | | | |
|---|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | - | - | 1,572 | 1,685 | 1,318 | 1,297 | 1,230 | 1,204 |
| THR | - | - | 2,482 | 2,112 | 2,239 | 1,731 | 2,042 | 1,680 |

- Dla układu 1000 równań dla obu metod:

| Progi opłacalności implementacji równoległej [ms] | | | | | | | | |
|---|---|----|-------|-------|-------|-------|-------|-------|
| liczba wątków | 1 | | 4 | | 8 | | 12 | |
| metoda | G | GJ | G | GJ | G | GJ | G | GJ |
| OMP | - | - | 1,667 | 1,640 | 1,330 | 1,306 | 1,231 | 1,209 |
| THR | - | - | 1,762 | 1,824 | 1,441 | 1,497 | 1,385 | 1,350 |

Progi opłacalności dla różnych implementacji programu przedstawione na wykresie poniżej:



5. Wnioski

Różne technologie równoległe, jak OpenMP, MPI i biblioteka wątków w C++, wykazują odmienne właściwości wydajnościowe w zależności od rodzaju operacji i rozmiaru danych. W przypadku mniejszych zbiorów danych oraz niewielkiej ilości wątków, implementacje równoległe mogą generować dodatkowe koszty związane z synchronizacją i zarządzaniem wątkami, co często prowadzi do ograniczonych korzyści wydajnościowych. Jednakże, wraz ze wzrostem rozmiaru danych i liczby wątków lub procesów, przyspieszenie staje się bardziej zauważalne. Dla większych zbiorów danych oraz większej liczby wątków, korzyści z równoległego przetwarzania stają się wyraźniejsze. Metoda Gaussa-Jordana ze względu na dodatkowe obliczenia (sprowadzanie macierzy do postaci jednostkowej rozszerzonej nie górno-trójkątnej jak w metodzie Gaussa) wypada gorzej pod względem czasowym w testach dla obu implementacji, nawet jeśli postać jednostkowa jest łatwiejsza do odczytania wyników przez ludzkie oko dla komputera są to zbędne obliczenia które tylko zajmują czas i zasoby.