Programowanie Równoległe i Rozproszone
- Projekt -
Rozwiązywanie układów równań liniowych

Adrian Bączek, 299235

1. Opis problemu

Zagadnieniem podejmowanym w realizowanym projekcie jest rozwiązywanie układów równań liniowych z zastosowaniem stworzonych do tego metod numerycznych.

2. Opis funkcjonalności programu

Przygotowany program umożliwiać będzie następujące czynności :

- Pobieranie danych wejściowych z pliku
- Proste walidowanie argumentów wywołania programu oraz zaimportowanych do niego danych
- Wykonywanie w sposób równoległy obliczeń na podstawie zaimportowanych danych w celu znalezienia rozwiązań układu równań
- Rozwiązywanie układu równań za z uwzględnieniem metody podanej jako argument wywołania programu
- Wypisywanie rozwiązania układu równań do pliku określonego jako jeden z argumentów wywołania

3. Specyfikacja techniczna stanowiska

System operacyjny	Windows 10 Professional v.10.0.19042.985
Procesor	Intel® Core™ i5-3230M CPU @2.60GHz
Język programowania	Rust
Edytor tekstowy	Visual Studio Code
Kompilator	rustc v. 1.52.1 (z dn. 2021-05-09)

4. Opis uruchomienia programu

Uruchomienie programu składać się będzie z wywołania 2 komend wskazanych poniżej.

Komenda kompilacyjna

```
D:\Studia\Sem6\PRIR\Lab[Projekt]\src>rustc main.rs
```

Komenda wywołująca program

Polecenie uruchomienia programu przybiera poniższą strukturę

“.\main.exe współczynniki wektor_prawej_strony liczba_wątków max_liczba_iteracji nazwa_pliku_wyjściowego metoda_rozwiązania”

przy czym,

współczynniki - ścieżka do pliku ze współczynnikami przy niewiadomych
wektor_prawej_strony - ścieżka do pliku z wektorem prawej stron
liczba_wątków - liczba wątków dzięki którym ma zostać znalezione rozwiązanie
metoda_rozwiązania - metoda jaką powinien być rozwiązany układ równań, możliwe wartości (uwzględniając wielkość liter) tego argumentu to “jacobi” oraz “gauss”.

Przykładowa komenda uruchomieniowa

```
D:\Studia\Sem6\PRIR\Lab[Projekt]\src>.\main.exe ../tests/test_package_13/coefficients.txt
../tests/test_package_13/y_values.txt 4 50 results_test_13 jacobi
```

5. Opis realizacji problemu

Podejście do postawionego problemu składa się z 2 elementów - odpowiedniego podziału pracy pomiędzy uruchomione wątki oraz ich synchronizację.

Dystrybucja zakresu pracy

Praca wątków polega na naprzemiennym aktualizowaniu przybliżenia rozwiązania układu równań. Decyzja o tym, który wątek ma aktualizować konkretną zmienną zależy od jego identyfikatora (przyjmującego wartości od 0 do n-1, gdzie n to liczba wątków). Koncept przydziału danych dla wątków można zaprezentować graficznie w poniższy sposób (na przykładzie układu równań z 10 niewiadomymi rozwiązywanego z użyciem 4 wątków) :

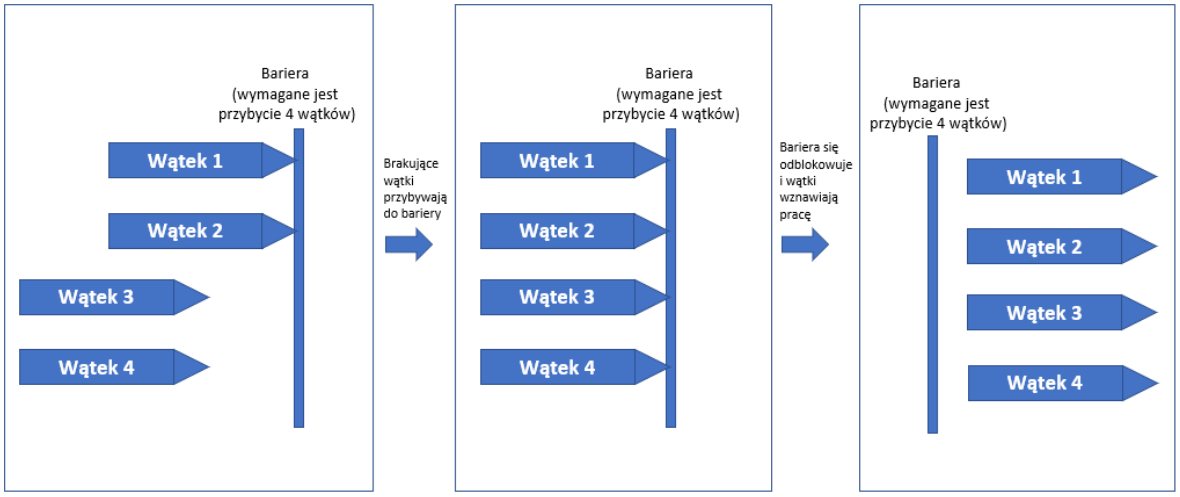
Graf. 1 - graficzne przedstawienie strategii podziału obliczanych wartości zmiennych (tutaj 10 - przykład) pomiędzy wątki (tutaj 4 - przykład)

W powyższym przykładzie wątki będą wyznaczać rozwiązania dla zmiennych o numerach odpowiednio :

Identyfikator wątku (i ozn. kolor)	Aktualizowane zmienne
0 (ozn. kolorem czerwonym)	1, 5, 9
1 (ozn. kolorem żółtym)	2, 6, 10
2 (ozn. kolorem zielonym)	3, 7
3 (ozn. kolorem niebieskim)	4, 8

Koncepcja synchronizacji wątków

Analizując zastosowane w projekcie strategie synchronizacji wątków trzeba przyznać, że są one odmienne dla każdej metody.
Niemniej posiadają one bardzo duży wspólny mianownik - wszystkie z nich bazują na koncepcji tzw. “barier”. Jej głównym założeniem jest istnienie bariery przy której działające wątki zatrzymują swoje działanie i czekają aż przybędzie do niej wystarczająca liczba wątków, po czym praca każdego wątku jest wznawiana.



Graf. 2 - graficzne przedstawienie koncepcji bariery

W implementacji metody Jacobiego wykorzystanie bariery wątkowej było konieczne wyłącznie przed rozpoczęciem kolejnej iteracji. Gwarantowało to dostępność wszystkich wyników w poprzedniej iteracji oraz bardzo dużą szansę na przybliżenie się do potencjalnego wyniku (pewnej liczby niebędącej NaN).
Przyglądając się z kolei implementacji metody Gaussa-Seidla, zauważyć trzeba, że zastosowanie bariery wątkowej nie wystarczy. Tutaj potrzebna była gwarancja, że wszystkie niezbędne do dalszych obliczeń wyniki będą dostępne. W tym celu wprowadzona została dodatkowa pętla while, w której wątek nie wykonywał żadnych innych operacji jak sprawdzanie czy są już znane konkretne dane.

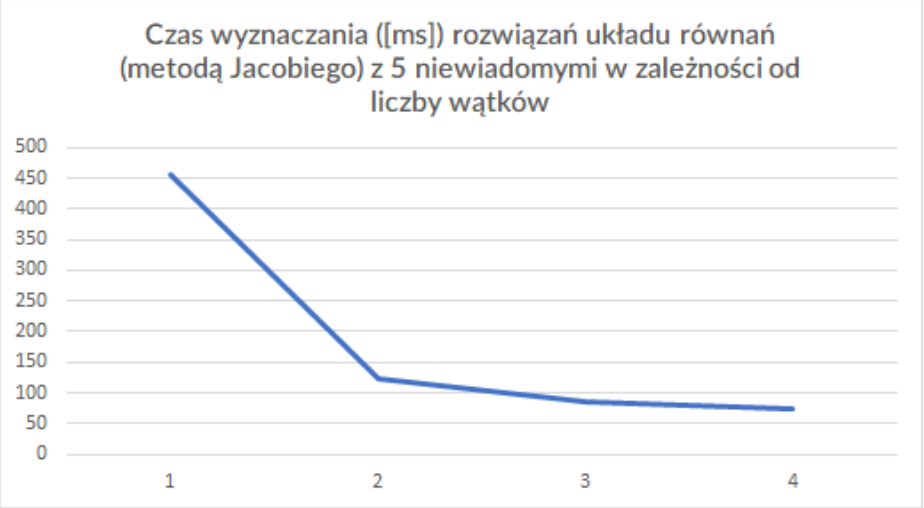
Porównując wskazane powyżej pomysły na wielowątkowe implementacje podanych metod trzeba przyznać, że bardziej wydajna okaże się być metoda Jacobiego (mimo lepszej zbieżności metody Gaussa-Seidla), ze względu na brak konieczności oczekiwania na niezbędne wyniki z aktualnej iteracji.
Poza zastosowaniem mechanizmu barier, dodatkowym synchronizatorem wątków są znane każdemu zamki (ang. mutexy).
Optymalizacją wykonywanych obliczeń (również uwzględnioną przez twórców tych metod) jest tutaj wyznaczanie wartości błędu rozwiązania
i porównywanie go z tolerowanym poziomem.

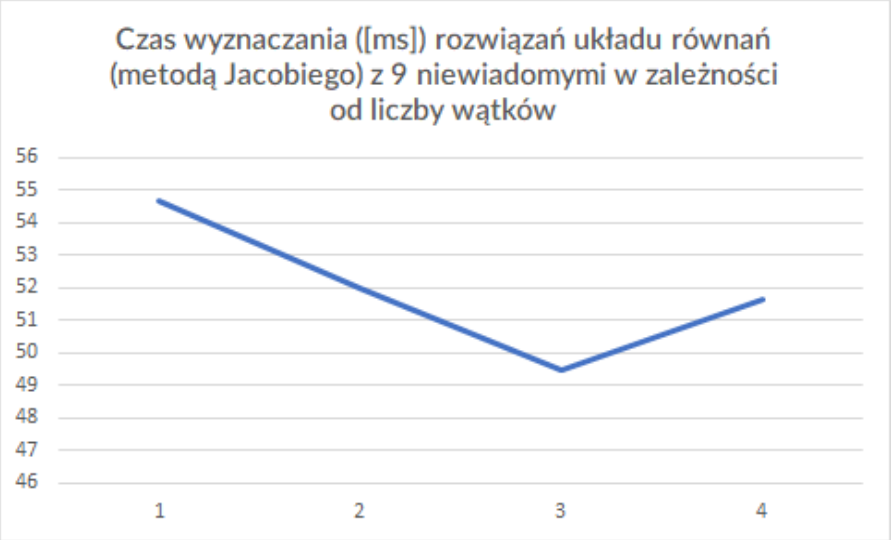
6. Rezultaty działania programu

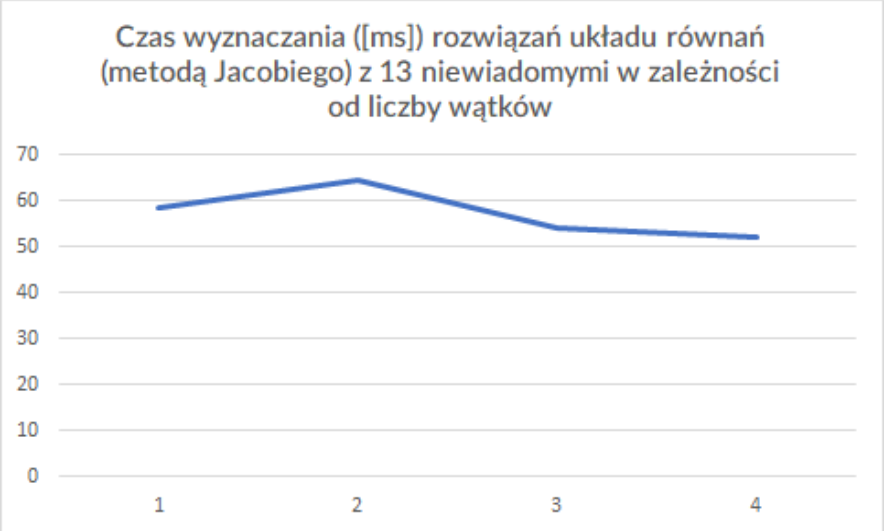
Zgodnie z przyjętymi na początku projektu założeniami, zaimplementowane rozwiązanie zostało przetestowane za pomocą wygenerowanych automatycznie danych testowych różnych rozmiarów. Liczebności niewiadomych w układzie równań należały do zbioru {5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29}, co powinno zweryfikować skalowalność programu.
Dla poszczególnych metod rezultaty prezentują się następująco :

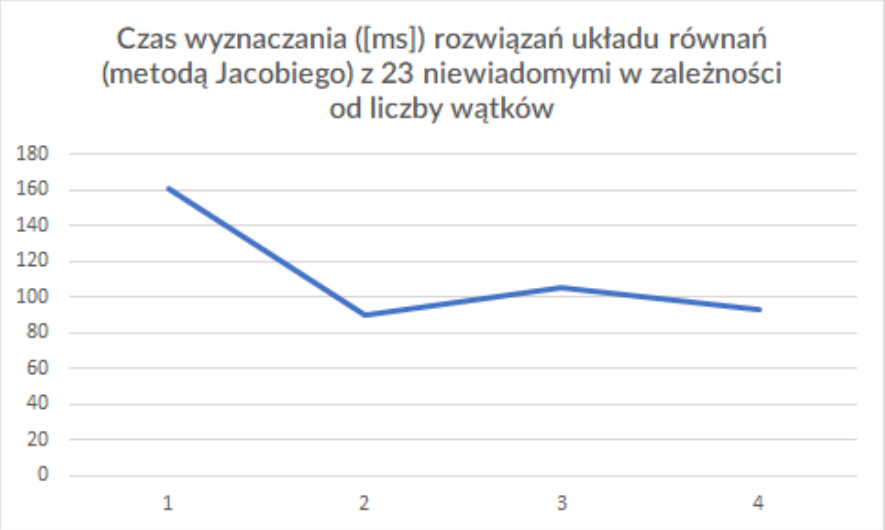
Metoda Jacobiego

Jak zostało wyżej wspomniane, każda metoda została zbadana za pomocą pakietów testów (13 pakietów). Poniżej umieszczone zostały niektóre wykresy, pokazujące osiągnięte przyspieszenie wynikające ze zwiększenia liczby wątków.







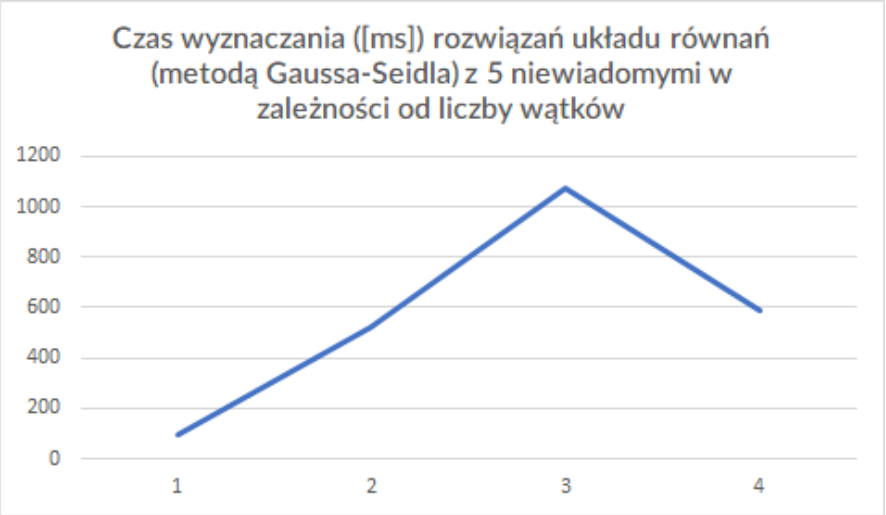


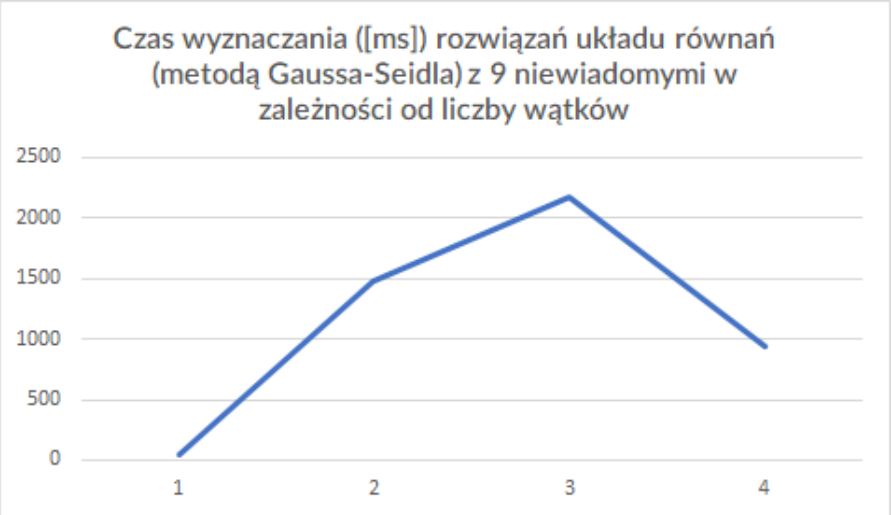
Spoglądając na powyższe wykresy, można stwierdzić, że program zachowuje się w sposób pożądany - osiągane jest przyspieszenie (niestety nie idealnie proporcjonalne do liczby wątków). Niemniej gdy spojrzy się na wszystkie stworzone wykresy (znajdujące się w pliku “[PRIR]Projekt_wyniki.xlsx”),

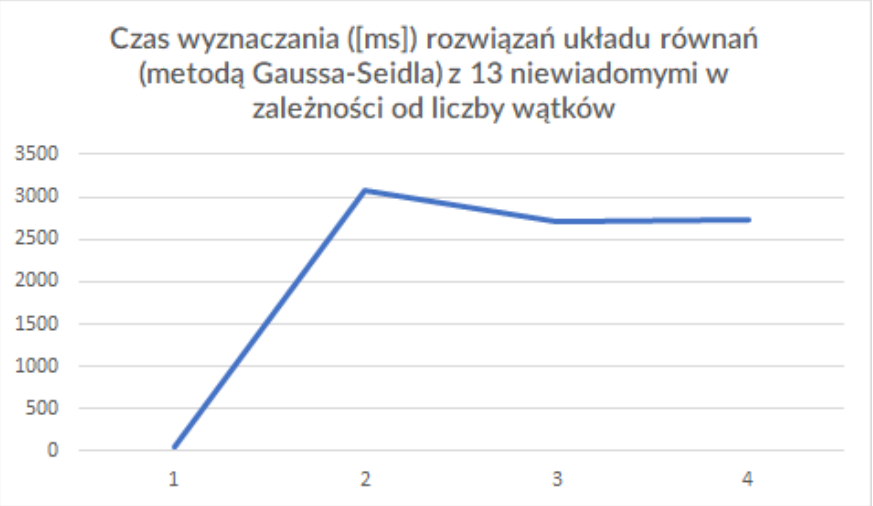
niestety nie można tego od razu powiedzieć. W niektórych przypadkach zwiększenie liczby wątków powodowało wydłużenie działania programu. Czym zatem mogło być to spowodowane? Prawdopodobnym jest to, że w trakcie wykonywania obliczeń przez wątki, niektóre z nich mogły być zatrzymane przez pewne przerwanie o wyższym priorytecie niż uruchomiony program - wówczas tłumaczyłoby to w pełni występujące rozbieżności między oczekiwaniami a rezultatami.

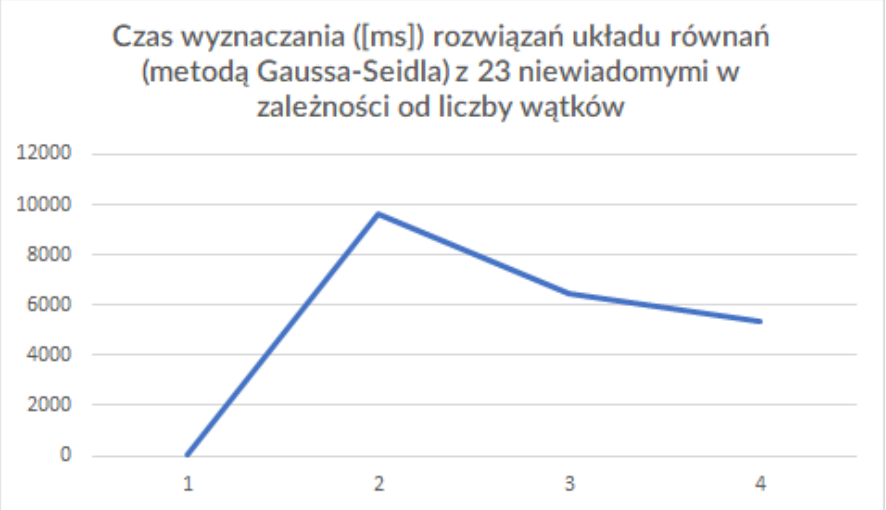
Metoda Gaussa-Seidla

Implementacja tej metody została przetestowana analogicznie jak omówiona powyżej metoda Jacobiego. Przebiegi czasu wykonywania obliczeń w zależności od liczby wątków przedstawiają się następująco :









Widok pokazanych powyżej przebiegów nie napawa entuzjazmem. Przyspieszenie nie zostało osiągnięte, co więcej większa liczba wątków przełożyła się tutaj na wydłużenie czasu wykonywania obliczeń.
Wynikać to może przede wszystkim ze wspomnianego oczekiwania na niezbędne przybliżenia rozwiązania. Pomimo pierwszego rozczarowania warto przyjrzeć się uważniej. W zdecydowanej większości przebiegów występuje tendencja malejącego czasu działania programu przy rosnącej liczbie wątków, lecz gdy spojrzy się na fragment od 2 do 4. Czas działania maleje, ale jest zawsze większy niż czas działania sekwencyjnego programu. Cóż z tego może wynikać? Wnioski z tego płynące mogą być takie - albo metoda Gaussa-Seidla jest trudna do zrównoleglenia (uściślając - trudniejsza w efektywnej synchronizacji wątków), albo też przyjęta strategia zrównoleglania zawierała detaliczne błędy, które zaważyły o takim a nie innym działaniu programu.

7. Projektowe wnioski końcowe

- Rozwiązywanie układów równań zawsze będzie obecne we wszelkiego rodzaju problemach informatyki
- Niektóre iteracyjne metody rozwiązywania układów równań są prostsze w zrównoleglaniu (metoda Jacobiego), inne natomiast są efektywniejsze gdy działają w trybie sekwencyjnym (metoda Gaussa-Seidla)
- Niektóre nowsze języki programowania, w momencie pisania programu wielowątkowego wymagają bardziej kreatywnego podejścia do kodowania (chociażby w aspekcie synchronizacji wątków i ich dostępu do danych).
- Możliwe jest stworzenie niskopoziomowego języka programowania, który nie wymaga od użytkownika ręcznego zarządzania pamięcią (przykładem jest tutaj język tego projektu - Rust)

Link do repozytorium na GitHub: https://github.com/ElAdriano/parallel-solver