Sprawozdanie
Rozwiązywanie układów równań liniowych (Skala)

Spis treści

Wstęp

Rozwiązywanie układów równań liniowych jest jednym z kluczowych problemów matematycznych, których rozwiązanie jest niezbędne w wielu dziedzinach, takich jak fizyka, inżynieria, ekonomia czy informatyka. Jednakże, obliczenia dla dużych układów równań liniowych, mogą być bardzo kosztowne i wymagają wiele czasu. Dlatego też metody numeryczne, takie jak metoda Gaussa, metoda Jacobiego, czy faktoryzacja LU, zostały opracowane, aby uprościć ten proces.

W ostatnich latach metody współbieżne stały się coraz bardziej popularne w dziedzinie obliczeń równoległych, umożliwiając szybsze i bardziej wydajne rozwiązywanie układów równań liniowych. W metodach tych, równania są podzielone na mniejsze pod problemy, które mogą być rozwiązane równolegle przez wiele procesorów lub rdzeni. Jest to szczególnie korzystne dla dużych układów równań liniowych.

W tej pracy zostanie omówiony projekt dotyczący rozwiązania układów równań liniowych w sposób współbieżny, z wykorzystaniem języka programowania Scala. Przede wszystkim zostaną przedstawione dwie metody: metoda Gaussa-Seidela oraz metoda Gaussa-Jordana. Metody zostały zaimplementowane z wykorzystaniem frameworka Akka.

Akka jest biblioteką w języku scala służącą do zrównoleglania i rozpraszania aplikacji. Podstawową jednostką jest aktor, który jest procesem w maszynie wirtualnej java. Aktor może otrzymywać wiadomości oraz wysyłać je do innych aktorów oraz wykonywać zadania. Aktor z punktu widzenia aplikacji nadzorującej znajduje się w systemie aktorów. Akka umożliwia również łatwe rozproszenie w ramach wielu komputerów (nie zrobiłem tego 🙁) dzięki wykorzystaniu możliwości tworzenia klastrów i wierzchołków (node).

Celem projektu była analiza możliwości biblioteki Akka w języku Scala w celu sprawdzenia działania oraz potencjalnych zastosowań w tworzeniu aplikacji równoległych i rozproszonych(skupiłem się głównie na wielu procesach).

Algorytm Gaussa-Jordana

Algorytm Gaussa-Jordana jest używany do rozwiązywania układów równań liniowych i znajdowania odwrotności macierzy. Działanie algorytmu opiera się na przeprowadzaniu operacji elementarnych na macierzach.

Algorytm rozproszony Gaussa-Jordana

W celu przeprowadzenia badania przygotowałem algorytm:

Dane początkowe:

1. Macierz AB – macierz połączoną macierzy wejściowej A i macierzy wynikowej B o rozmiarze n x n+1
2. M – maksymalna liczba wierszy w bloku

Algorytm:

1. Podziel wiersze macierzy AB na n bloków po maksymalnie m i utwórz dla każdego bloku aktora
2. Dla każdego wiersza o indeksie i w macierzy AB wykonaj:
   A. Znajdź wiersz o elemencie o maksymalnej wartości bezwzględnej w każdym aktorze dla kolumny i i indeksie j wiersza większym niż indeks i (równolegle) i wybierz element o indeksie k o największej wartości spośród nich
   B. Zamień wiersz i z k

   C. Podziel wiersz o indeksie i tak żeby na diagonali znajdowała się 1
   D. Dla każdego bloku roześlij wiersz o indeksie i i odejmij go od każdego wiersza (równolegle)
3. Pobierz wartość ostatniej kolumny z każdego bloku (równolegle) połącz macierz x i zakończ algorytm

Wynik działania algorytmu:

1. Macierz wynikowa x

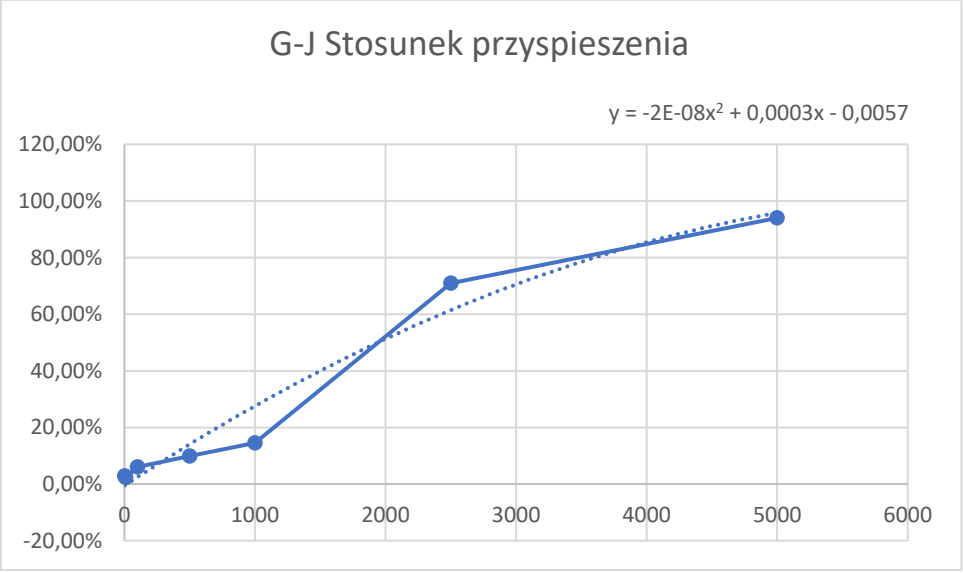
Opracowany w ten sposób algorytm wykonuje dużą część zadań w sposób równoległy. Umożliwia on również rozproszenie algorytmu korzystając z Akka umożliwia to przede wszystkim zmniejszenie ilości wymaganej pamięci do przeprowadzenia obliczeń w ramach 1.

Działanie algorytmu

Wyniki działania algorytmu są dla mnie nie do końca takie jak oczekiwałem. Czas wykonania jest sporo dłuższy niż w przypadku wykonywania nawet dla dużych macierzy, jednak widać że czas potrzebny ma wykonania rośnie szybciej w przypadku nie zrównoleglonej funkcji.

Wyniki sumaryczne Gauss-Jordan			
n	standardowe [ms]	najlepszy czas dla implementacji akka [ms]	stosunek
3	10	353	2,83%
10	9	386	2,33%
100	27	449	6,01%
500	58	588	9,86%
1000	138	951	14,51%
2500	6110	8609	70,97%
5000	63360	67414	93,99%

Jeśli spojrzymy na wykres stosunku czasów działania programu i zastosujemy funkcję trendu (wielominowa 2 stopnia) to możemy uzyskać wzór mówiący o szybkości wzrostu stosunku w tym przedziale. Widać, że szybkość rozwiązania macierzy rośnie dość szybko, lecz wykres tego wzrostu przybiera kształt logarytmiczny.



Analiza wyników

W poniższych tabelach zaprezentowałem uśrednione wyniki czasowe dla kilku iteracji dla zrównoleglonego algorytmu Gaussa-Jordna.

Gauss-Jordan Akka 3x3		
m	liczba aktorów	czas wykonania [ms]
1	3	353

Gauss-Jordan Akka 10x10		
m	liczba aktorów	czas wykonania [ms]
10	1	386

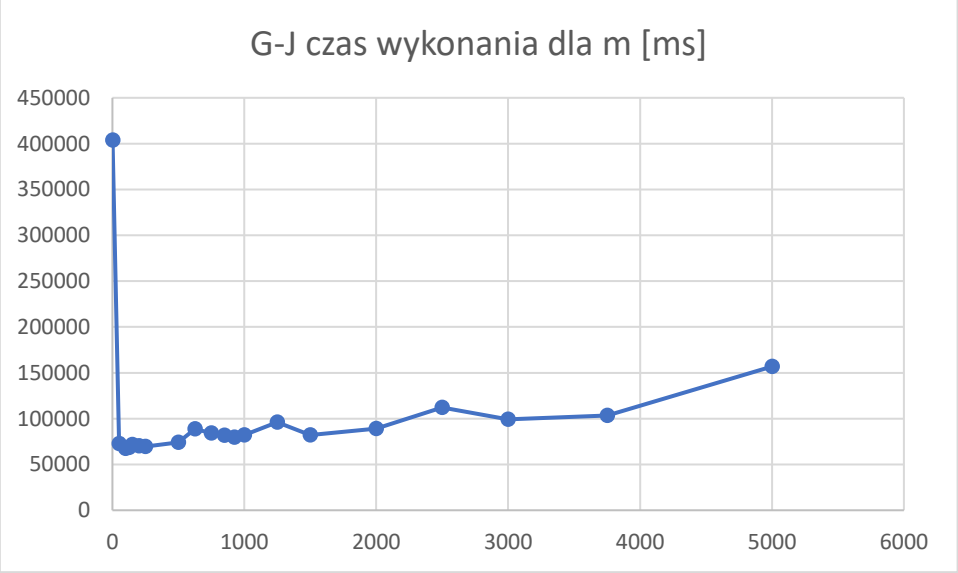
Gauss-Jordan Akka 100x100		
m	liczba aktorów	czas wykonania [ms]
25	4	449

Gauss-Jordan Akka 500x500		
m	liczba aktorów	czas wykonania [ms]
375	2	588

Gauss-Jordan Akka 1000x1000		
m	liczba aktorów	czas wykonania [ms]
100	10	951

Gauss-Jordan Akka 2500x2500		
m	liczba aktorów	czas wykonania [ms]
175	15	8609

Gauss-Jordan Akka 5000x5000		
m	liczba aktorów	czas wykonania [ms]
1	5000	403648
50	100	72531
100	50	67414
125	40	68526
150	34	71750
200	25	70116
250	20	69443
500	10	73957
625	8	88834
750	7	84270
850	6	81652
925	6	79590
1000	5	82184
1250	4	96187
1500	4	81961
2000	3	88954
2500	2	111968
3000	2	99125
3750	2	103502
5000	1	156775



Na podstawie uzyskanych wyników zaobserwowałem, że wielkość klastra m jest odwrotnie proporcjonalna do ilości wierszy w tabeli. Jest to dla mnie dość zaskakujący wynik mogący sugerować, że w czas potrzebny na przeprowadzenie operacji oraz wysłanie i przetworzenie wiadomości przy takich założeniach równoważą się.

Dodatkowo można zaobserwować że w przypadku wartości skrajnych nie są ono optymalne i prowadzą do dłuższego czasu wykonywania się programu.

Algorytm Gaussa Seidla

Algorytm Gaussa-Seidla jest jednym z popularnych iteracyjnych algorytmów rozwiązywania układów równań liniowych. Wykorzystuje on wartości niewiadomych z poprzednich iteracji do aktualizacji wartości w kolejnych iteracjach, co prowadzi do stopniowego zbliżania się do rozwiązania ostatecznego.

Algorytm rozproszony Gaussa-Seidla

Dane początkowe:

1. Macierz A – macierz wejściowa o rozmiarze n x n
2. Macierz B – macierz wynikowa o rozmiarze n x 1
3. M – maksymalna liczba wierszy w bloku
4. Eps – minimalna wymagana dokładność – zmiana wartości
5. Iteration – maksymalna liczba iteracji
6. Wektor wynikowy x – Wektor o rozmiarze n x 1 początkowo wypełniony 0

Algorytm:

1. Podziel wiersze macierzy A na n bloków po maksymalnie m i utwórz dla każdego bloku aktora
2. Dopóki iteracja < Iteration i zmiana dokładności < eps wykonuj (równolegle) i brak sygnału o końcu od innych aktorów
   A. Wylicz nowy podwektor x ze wzoru dla każdego wiersza m (wzór z algorytmu gaussa-seidla)
   B. Wyślij wyliczone nowe wartości podwektora x (Tylko dla naszego podzbioru) do ostatniego wektora na liście. (Jeśli ostatni wektor zbierz informacje o podwektorach x, połącz w wektor x i odeślij do innych aktorów)
   C. Czekaj na informacje o nowym wektorze x
3. Jeśli nie iteracja < Iteration lub nie zmiana dokładności < eps roześlij informacje o końcu do innych aktorów.
4. Jeśli aktor o indeksie 0 zwróć macierz wynikową X i zakończ algorytm

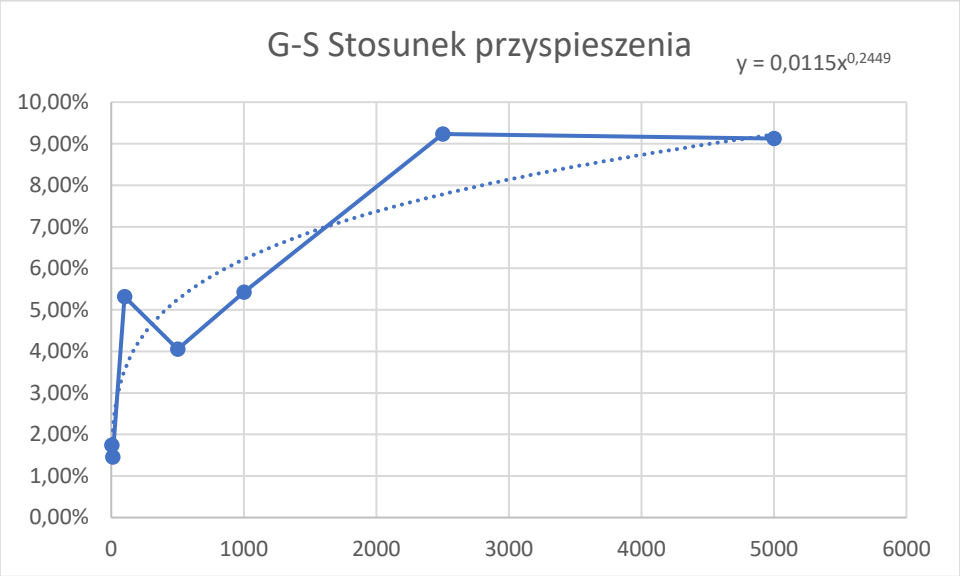
Wynik działania algorytmu:

1. Macierz wynikowa X

Działanie algorytmu

Algorytm działa w sposób dla mnie rozczarowujący. Czas przyspieszenia jaki można uzyskać w dłuższej perspektywie jest wysoce niesatysfakcjonujący i nie dający żadnych podstaw do myślenia o potencjalnych uzyskach. Pomimo wielokrotnych prób poprawy tego stanu rzeczy nie udało mi się tego poprawić. Czas wykonania algorytmu jest kilkukrotnie dłuższy. Od wersji nie rozproszonej / zrównoleglonej.

Wyniki sumaryczne Gauss-Seidla			
n	standardowe [ms]	najlepszy czas dla implementacji akka [ms]	stosunek
3	6	344	1,74%
10	5	344	1,45%
100	20	376	5,32%
500	19	469	4,05%
1000	26	479	5,43%
2500	65	704	9,23%
5000	140	1534	9,13%



Przyczyn takiego stanu rzeczy mógłbym doszukiwać się głównie z powodu komunikacyjnego. Aktor po każdej iteracji musi zebrać wszystkie podwektory x, przesłać do aktora o ostatnim indeksie po czym aktor końcowy musi złożyć wektor x i rozesłać go do wszystkich pozostałych aktorów. Próbowałem rozwiązać ten problem na wiele sposobów wykorzystując różne sposoby komunikacji (np. wektor wysyłał odpowiedzi 0-> 1->2-> … -> n-> n-1) jednak skok wydajności nie był duży. Rozwiązaniem mogłoby być zastosowania grupowania opartego na metodzie dziel i zwyciężaj jednak takie podejście było dla mnie dość ciężkie od zaimplementowania (Może rozwiązaniem było odwrócenie komunikacji ? nie wysyłanie do aktora tworzącego nową macierz x, a pobieranie podwektorów od pozostałych ? – potencjalne ryzyko spowolnienia jeszcze większego – czekanie na odpowiedź – konieczność synchronizacji).

Analiza wyników

Gauss-Seidl Akka 3x3		
m	liczba aktorów	czas wykonania [ms]
3	1	344

Gauss-Seidl Akka 10x10		
m	liczba aktorów	czas wykonania [ms]
5	2	344

Gauss-Seidl Akka 100x100		
m	liczba aktorów	czas wykonania [ms]
25	4	376

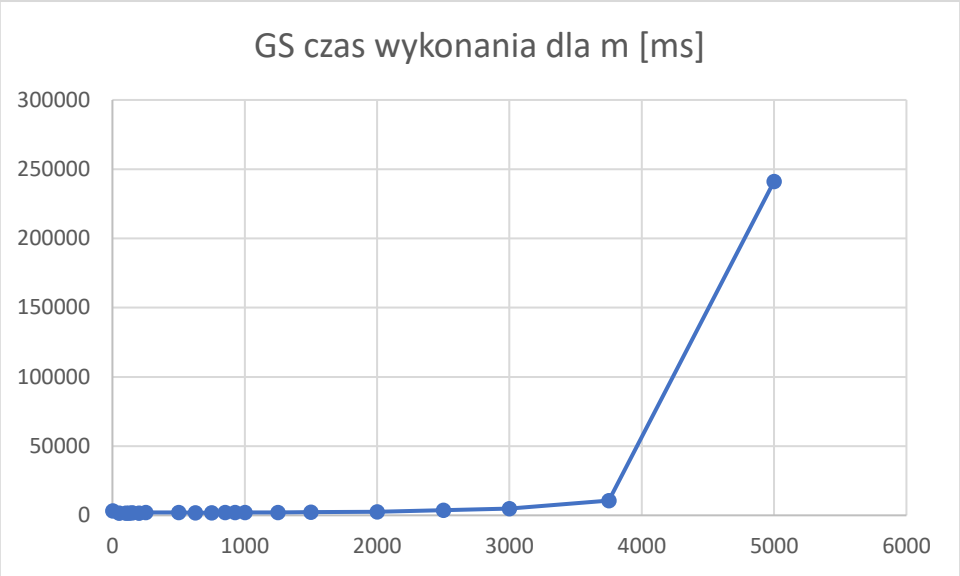
Gauss-Seidl Akka 500x500		
m	liczba aktorów	czas wykonania [ms]
300	2	469

Gauss-Seidl Akka 1000x1000		
m	liczba aktorów	czas wykonania [ms]
75	14	479

Gauss-Seidl Akka 2500x2500		
m	liczba aktorów	czas wykonania [ms]
100	25	704

Gauss-Seidl Akka 5000x5000		
m	liczba aktorów	czas wykonania [ms]
1	5000	3222
50	100	1535
100	50	1534
125	40	1551
150	34	1775
200	25	1659
250	20	2117
500	10	2011
625	8	1946
750	7	1911
850	6	2099
925	6	2079
1000	5	2217
1250	4	2194
1500	4	2389
2000	3	2653
2500	2	3709
3000	2	4827
3750	2	10580
5000	1	241222

Na podstawie zebranych wyników mogę stwierdzić że w przypadku mojego algorytmu głównym problemem jest odbieranie danych o dużej wielkości (Pamięć). Świadczą o tym czasy wykonania, dla których punkt optimum znajduje się w przedziale wielkości m 75-100 wierszy.



Podsumowanie

Z przeprowadzonych testów wynika, że zaprojektowany przeze mnie algorytm rozproszony/zrównoleglony Gaussa Jordana umożliwia przyspieszenie macierzy o dużej wielkości w przeciwieństwie do napisanego i zaprojektowanego algorytmu zrównoleglonego/rozproszonego Gaussa Seidla.

Pomimo oczywistych wad 2 algorytmu działa on i tak zdecydowanie szybciej od algorytmu GJ. W związku z tym nie jest on całkowicie skazany na porażkę w przypadku, gdyby okazało się że nie można go przyspieszyć. Oba algorytmy można przy niewielkich modyfikacjach wykorzystać w celu rozproszenia na wiele komputerów – jednostek komputerowych co umożliwiłoby obliczanie większych układów macierzowych – np. macierzy 10.000x10.000 (w moim komputerze brakuje pamięci ram do jego obliczenia). Dzięki temu taki algorytm zyskiwałby na szybkości dzięki temu, że algorytm nie musiał by wymieniać pamięci ram na pamięć z pamięcią dysku twardego. W takiej hipotetycznej sytuacji algorytm ten mógłby okazać się wielokrotnie szybszy od swojej nie zrównoleglonej wersji.

Podczas wykonywania tego ćwiczenia dowiedziałem się wielu interesujących rzeczy z zakresu trudności projektowania i budowy algorytmów rozproszonych i zrównoleglonych, potencjalnych problemach optymalizacyjnych takich algorytmów oraz dowiedziałem się podstaw języka programowania Scala wraz z pewnymi podstawami na temat scala i akka. Mam nadzieje, że uzyskana przeze mnie wiedza pomoże mi w przyszłości.

Github - https://github.com/ciszewsj/Wspolbierzne/tree/main/projekt

Z wyrazami szacunku,

311192 Ciszewski Jakub