Aplikacja rozproszona RMI rozwiązująca układ równań liniowych z wykorzystaniem dekompozycji LU

1. Cel projektu

Celem projektu było wykonanie aplikacji umożliwiającej rozwiązywanie układu równań liniowych. Do znalezienia wektora rozwiązań układu równań program miał wykorzystywać rozkład LU, którą powinien sam generować na podstawie początkowych danych układu.

Podstawowym założeniem projektowym stało się zastosowanie w tym przypadku obliczeń rozproszonych na minimum dwóch serwerach.

2. Analiza problemu matematycznego

Rozpatrujemy układ równań:

$$Ax = b$$

zapisany w postaci macierzowej:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

Macierz współczynników A oraz wektor wyrazów wolnych są dane, natomiast wartościami szukanymi są elementy wektora niewiadomych x.

2.1. Dekompozycja LU

Macierz A możemy zapisać jako iloczyn:

$$A = L \cdot U$$

Gdzie macierze L oraz U są zdefiniowane w sposób ogólny następująco:

$$L = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ l_{2,1} & 1 & 0 & \dots & 0 & 0 \\ l_{3,1} & l_{3,2} & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ l_{n-1,1} & l_{n-1,2} & l_{n-3,3} & \dots & 1 & 0 \\ l_{n,1} & l_{n,2} & l_{n,3} & \dots & l_{n,n-1} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,1} & \dots & u_{1,n-1} & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \dots & u_{2,n-1} & u_{2,n} \\ 0 & 0 & u_{3,3} & \dots & u_{3,n-1} & u_{3,n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & 0 & \dots & 0 & u_{n,n} \end{bmatrix}$$

Przedstawiony poniżej algorytm rozdzielania macierzy A na L oraz U działa „w miejscu” – po jego zakończeniu macierze L oraz U będą się znajdować w tym samym miejscu w pamięci, gdzie wcześniej była przechowywana macierz A. W macierzy L istotna jest tylko część, znajdująca się poniżej głównej przekątnej. Elementy, znajdujące się na niej wynoszą zawsze 1. Macierz U zawiera natomiast ważne elementy na głównej przekątnej i ponad nią. Macierz A’, powstała z A, wygląda następująco:

$$A' = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \dots & u_{3,n-1} & u_{1,n} \\ l_{2,1} & u_{2,2} & u_{2,3} & \dots & u_{3,n-1} & u_{2,n} \\ l_{3,1} & l_{3,2} & u_{3,3} & \dots & u_{3,n-1} & u_{3,n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ l_{n-1,1} & l_{n-1,2} & l_{n-1,3} & \dots & u_{n-1,n-1} & u_{n-1,n} \\ l_{n,1} & l_{n,2} & l_{n,3} & \dots & l_{n,n-1} & u_{n,n} \end{bmatrix}$$

Poniżej znajduje się wersja zorientowana kolumnowo algorytmu rozkładu LU macierzy w miejscu.

```
Dla każdej kolumny macierzy A k=1:n-1 wykonuj:
    # normalizacja kolumny k
    Dla każdego wiersza macierzy A s=k+1:n wykonuj:
        a[s][k] = a[s][k] / a[k][k]
    # modyfikacja elementów, które leżą równocześnie poniżej
    # kolumny j oraz poniżej wiersza i
    Dla każdej kolumny macierzy A j=k+1:n wykonuj:
        Dla każdego wiersza macierzy A i=k+1:n wykonuj:
            a[i][j] = a[i][j] - a[j][k]*a[k][j]
```

2.2. Układ równań liniowych

Po rozłożeniu macierzy A na macierze L oraz U, układ równań liniowych można zapisać jako:

$$L \cdot U \cdot x = b$$

Wektor niewiadomych może zostać obliczony w dwóch krokach:

$$\begin{cases} L \cdot y = b \\ U \cdot x = y \end{cases}$$

Dzięki tej metodzie rozwiązanie układu równań jest prostsze, gdyż postać macierzy L i U pozwala stosunkowo łatwo wyliczać wartości wektorów y oraz x. Algorytm składa się z dwóch części:

1. podstawienie w przód:

$$y_1 = b_1$$

$$y_i = b_i - \sum_{j=1}^{i-1} l_{i,j} y_j \ , \ dla\ i = 2,3, \dots, n$$

2. podstawienie wstecz:

$$x_n = \frac{y_n}{u_{n,n}}$$

$$x_i = \frac{1}{u_{i,i}}\left(y_i - \sum_{j=i+1}^{n} u_{i,j}x_j\right), \quad dla\ i = n-1, n-2, \ldots, 1$$

Aby zaoszczędzić miejsca w pamięci, wektor pomocniczy y można przechowywać w tym samym miejscu, co wektor x., gdyż podczas ich obliczania żadne potrzebne dane nie zostaną nadpisane przed ich użyciem.

3. Struktura projektu

Projekt składa się z dwóch części: głównej, napisanej w języku Java oraz kodu C, realizującego najbardziej czasochłonne operacje. Struktura katalogów odzwierciedla ten podział w celu łatwiejszego zrozumienia kodu.

W części napisanej w Javie można wyróżnić:

- klienta, realizującego wczytanie danych układu równań z podanych plików, wykonywującego zapytania do serwerów i zapisującego wyniki obliczeń;
- serwera dekompozycji, wykonującego dekompozycję LU macierzy współczynników;
- serwera rozwiązywania, który rozwiązuje przysłany mu układ równań i odsyła klientowi rozwiązanie.

W skład projektu wchodzą następujące pliki:

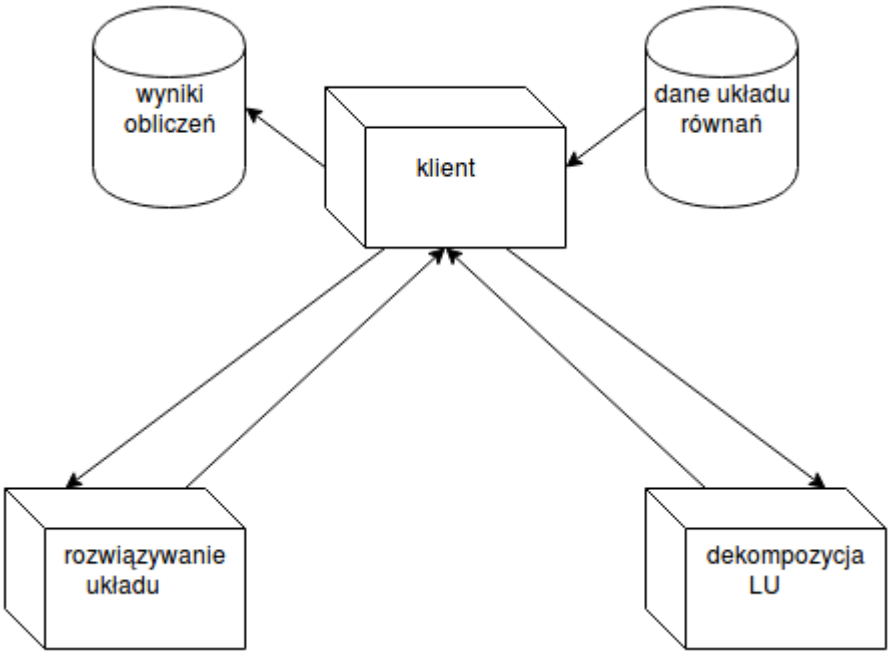
- */resources/*
  - *A.txt* – macierz współczynników A_1 o wymiarach 400 x 400;
  - *A2.txt* – macierz współczynników A_2 o wymiarach 5 x 5;
  - *A3.txt* – macierz współczynników A_3 o wymiarach 1000 x 1000;
  - *b.txt* – wektor b_1 wyrazów wolnych równania dla macierzy A_1, o wymiarach 400 x 1;
  - *b2.txt* – wektor b_2 wyrazów wolnych równania dla macierzy A_2, o wymiarach 5 x 1;
  - *b3.txt* – wektor b_2 wyrazów wolnych równania dla macierzy A_2, o wymiarach 1000 x 1;
- */src/*
  - *clib/src/*
    - *decompose.c* – funkcje obsługujące dekompozycje macierzy współczynników do macierzy LU;
    - *Decomposer.c* – opakowanie na funkcję do dekompozycji, tworzące interfejs między Javą oraz C;
    - *solve.c* – funkcje obsługujące rozwiązywanie układu równań liniowych;
    - *Solver.c* – opakowanie na funkcję do rozwiązywania układu równań, tworzące interfejs między Javą oraz C;
  - *java/*
    - *client/*
      - *Client.java* – klient aplikacji, inicjujący oraz kontrolujący proces rozwiązywania układu równań;
    - *common/*
      - *DecomposeIface.java* – interfejs, definiujący jakie metody musi posiadać serwer dekompozycji; potrzebne, aby klient wiedział, jaką metodę może wywołać zdalnie;

- *Equations.java* – klasa opisująca układ równań; zawiera macierz współczynników, wektor niewiadomych oraz wektor wyrazów wolnych; jej obiekty mogą być serializowane;
- *Matrix.java* – klasa do obsługi macierzy, używanych w programie, pozwala na odczyt danych z pliku i zapis do pliku, a także kontroluje dostęp do wartości ich elementów; jej obiekty mogą być serializowane;
- *MatrixLoaderException.java* – wyjątek zgłaszany w sytuacji, gdy macierz nie może być załadowana ze wskazanego pliku; problemem może być np. zły format użytych danych;
- *ParamException.java* – wyjątek zgłaszany, jeżeli przy odwołaniu się do pozycji w macierzy lub wektorze zostanie podana zła wartość – ujemna lub większa, niż rozmiar;
- *SolverIface.java* – interfejs, definiujący jakie metody musi posiadać serwer rozwiązywania układu równań; potrzebne, aby klient wiedział, jaką metodę może wywołać zdalnie;
- *Vector.java* – klasa do obsługi wektorów, używanych w programie, pozwala na odczyt danych z pliku i zapis do pliku, a także kontroluje dostęp do wartości ich elementów; jej obiekty mogą być serializowane;
- *VectorLoaderException.java* – wyjątek zgłaszany w sytuacji, gdy macierz nie może być załadowana ze wskazanego pliku; problemem może być np. zły format użytych danych;

- *.gitignore* – lista lokalizacji, które mają być ignorowane przez system kontroli wersji;
- *Makefile* – plik programu MAKE, który pozwala budować aplikację, w zależności od podanych ustawień;
- *README.md* – podstawowe informacje o projekcie;

4. Działanie aplikacji

Program umożliwia rozwiązywanie układu równań liniowych przy pomocy dekompozycji LU w sposób rozproszony na maksymalnie trzech różnych komputerach: kliencie i dwóch serwerach. Architektura aplikacji została przedstawiona na schemacie 1.



Schemat 1 Architektura aplikacji

Przed rozpoczęciem pracy obydwa serwery powinny być włączone oraz widoczne dla klienta. Dla każdego z nich można podać numer portu, na jakim będzie nasłuchiwał komunikacji.

Klient wymaga podania mu zarówno nazw hostów, jak też portów obydwu serwerów, których wymaga do poprawnego zrealizowania zadania. Po uruchomieniu ładuje on macierz współczynników oraz wektor wyrazów wolnych układu równań z podanych plików.

Pierwszą operacją, do wykonania której klient korzysta z serwera, jest dekompozycja LU macierzy A. W tym celu łączy się on z odpowiednim serwerem i wywołuje na nim zdalnie metodę *decompose()*. Serwer korzysta z kodu, napisanego w języku C, którego używa przy pomocy mechanizmu JNI. Funkcja natywna zwraca kod błędu, a wynikiem działania metody *decompose()* jest macierz LU.

Drugą operacją, przy której wykonania klient używa serwera jest rozwiązanie układu równań. Po połączeniu się z drugim serwerem wywołuje on na nim metodę *solve()*. On również używa kodu w języku C, uruchamiając go przy pomocy JNI. Funkcja natywna zwraca kod błędu, a wynikiem działania metody *solve()* jest rozwiązany układ równań liniowych.

Po otrzymaniu wyników od drugiego serwera klient zapisuje do pliku obliczony wektor niewiadomych i kończy pracę. Serwery natomiast są uruchomione w stanie nasłuchiwania do czasu, aż zostaną ręcznie wyłączone.

Taka architektura nie wymaga bardzo mocnego komputera klienta, gdyż bardziej skomplikowane operacje, czyli przede wszystkim dekompozycja LU, a także samo rozwiązanie układu równań mogą być wykonywane zdalnie na szybszych maszynach. Użytkownik aplikacji może uruchomić dwa serwery na wydajnych maszynach, dostępnych przez sieć internetową, a następnie używać ich do rozwiązywania układów równań na różnych, nawet dużo słabszych maszynach.

Zarówno klient, jak i serwery mierzą czas wykonywanych operacji. W przypadku klienta są to długość wykonywania się całego programu oraz obydwu operacji – dekompozycji i rozwiązywania układu równań.

5. Testowanie

Program został przetestowany w środowisku pracowni komputerowej dla dwóch układów równań. Do prezentacji wyników czasowych przyjęto następujące oznaczenia:

- T_C – całkowity czas uruchomienia klienta;
- Tc_{LU} – czas rozkładu LU, zmierzony przez klienta;
- Tc_S – czas rozwiązania układu równań, zmierzony przez klienta;
- Ts_{LU} – czas wykonania rozkładu LU przez serwer;
- Ts_S – czas rozwiązania układu równań przez serwer.

5.1. Mały układ równań

Jednym z przygotowanych przez nas układów był zawierający pięć równań. Poniżej są przedstawione macierz A oraz wektor b.

$$A = \begin{bmatrix} 25 & 94 & 10 & 54 & 64 \\ 60 & 32 & 74 & 12 & 39 \\ 62 & 5 & 97 & 46 & 38 \\ 51 & 20 & 84 & 67 & 15 \\ 6 & 41 & 78 & 94 & 38 \end{bmatrix}$$

$$b = \begin{bmatrix} 14 \\ 56 \\ 91 \\ 41 \\ 31 \end{bmatrix}$$

Program dokonał rozkładu macierzy A na dwie macierze: L oraz U.

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 2.4 & 1 & 0 & 0 & 0 \\ 2.48 & 1.178306 & 1 & 0 & 0 \\ 2.04 & 0.88719 & 1.448319 & 1 & 0 \\ 0.24 & -0.095248 & 6.049239 & 19.417806 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 25 & 94 & 10 & 54 & 64 \\ 0 & -193.6 & 50 & -117.6 & -114.6 \\ 0 & 0 & 13.284711 & 50.64876 & 14.313843 \\ 0 & 0 & 0 & -12.181996 & -34.619024 \\ 0 & 0 & 0 & 0 & 597.362202 \end{bmatrix}$$

Ostatecznym efektem jego działania było obliczenie szukanego wektora x:

$$x = \begin{bmatrix} 0.558364 \\ -1.062448 \\ -0.004057 \\ 0.196655 \\ 1.395816 \end{bmatrix}$$

Tabela 1 przedstawia informacje o tym, ile czasu zajęło programowi wykonanie zadania.

Czas	Wartość [ms]
T_C	333
Tc_{LU}	169
Tc_S	26
Ts_{LU}	< 1

Ts_S	< 1

Tabela 1 Wyniki czasowe wykonania aplikacji dla małego układu równań

5.2. Duży układ równań

Drugim układem równań, na którym testowaliśmy program, był układ 400 równań liniowych. Rozmiar macierzy A w tym przypadku wynosił 400 x 400.

Tabela 2 przedstawia informacje o tym, ile czasu zajęło programowi wykonanie zadania.

Czas	Wartość [ms]
T_C	3633
Tc_{LU}	1403
Tc_S	246
Ts_{LU}	967
Ts_S	7

Tabela 2 Wyniki czasowe wykonania aplikacji dla dużego układu równań

5.3. Bardzo duży układ równań

Ostatnim z rozwiązywanych układów był układ 1000 równań liniowych. Rozmiar rozwiązywanego problemu był na tyle duży, że znalazło to zauważalne odzwierciedlenie w czasach, poświęconych przez program na dokonanie rozkładu LU. Zostały one przedstawione w tabeli 3.

Czas	Wartość [ms]
T_C	24408
Tc_{LU}	15454
Tc_S	206
Ts_{LU}	15158
Ts_S	43

Tabela 3 Wyniki czasowe wykonania aplikacji dla bardzo dużego układu równań

6. Wnioski

Program poprawnie oblicza układ równań przy pomocy rozkładu LU macierzy współczynników. Zaprezentowane z tym sprawozdaniu wyniki opisują uruchomienie wszystkich trzech komponentów na jednym komputerze. Dużo większe korzyści działania programu można by zauważyć w sytuacji, gdy komputer klienta posiada małe zasoby. Wówczas rozwiązanie dużego układu może zająć ogromną ilość czasu. Jeżeli posiada on dodatkowo przynajmniej jeden serwer o większej wydajności, z dostępem do Internetu, wówczas może uruchomić na nim serwery naszej aplikacji. Dzięki użyciu lepszego sprzętu program będzie mógł obliczyć rozwiązanie w krótszym czasie. Oczywiście, możliwe byłoby również uruchomienie całego programu nierozproszonego na mocniejszym serwerze, jednak opcja, w której użytkownik może wykonać lokalnie program klienta, a ten automatycznie połączy się z serwerem i użyje jego zasobów, jest wygodniejsza dla korzystających z aplikacji. Dodatkowo program umożliwia uruchomienie serwerów jako usług na maszynach widocznych w sieci i wykorzystywanie ich przez wielu użytkowników, posiadających tylko klientów aplikacyjnych.

Analizując wyniki czasowe można zauważyć, iż czas dekompozycji LU albo czas rozwiązania układu równań są różne w zależności od tego, czy były mierzone na serwerze czy w kliencie. Pokazuje to, iż na długość wykonywania się tych czynności wpływ ma również prędkość komunikacji między klientem a serwerami. W celu uzyskania możliwie najlepszej wydajności należałoby zadbać o wystarczająco szybkie połączenie internetowe. Zawsze będzie występować pewna składowa czasu, pochodząca od procesu synchronizacji. Opłaca się jednak stosować takie rozwiązanie w sytuacji, gdy duży rozmiar problemu i szybszy sprzęt równoważą ten dodatkowy nakład czasu.

Porównanie osobno czasu trwania dekompozycji LU oraz rozwiązywania układu pokazuje, iż pierwsza z nich wykonuje się znacznie dłużej niż druga i różnica ta wzrasta wraz z rozmiarem problemu. Stąd można wnioskować, iż serwer *decomposer* wymaga znacznie większych zasobów niż serwer *solver*. Z tego względu przeniesienie go na osobną, znacznie szybszą maszynę, przyniesie wymierne skutki w postaci znaczącego wzrostu szybkości działania aplikacji.

W tabeli 4 przedstawiono czasy działania programu *LUdecoMPI* dla wersji nierównoległej, przy obliczaniu małego oraz dużego układu równań.

n	5	400
T [ms]	0.0315	43.1274

Tabela 4 Czas wykonania programu LUdecoMPI dla takich samych układów, jak użyte w niniejszym zadaniu

Jak można zaobserwować, czas wykonywania się tamtego programu, napisanego w języku C jest krótszy niż aplikacji rozproszonej. Porównanie całkowitego czasu mogłoby być mylące ze względu na dodatkowy narzut komunikacyjny. Jednak okazuje się, że nawet funkcje dekompozycji LU czy rozwiązywania układu równań, korzystające również z kodu natywnego, wykonują się wolniej, niż program napisany w czystym C.

7. Instrukcja działania programu

Po rozpakowaniu projektu wchodzimy do katalogu głównego projektu, w którym znajduje się plik makefile. Poniżej znajduje się opis zadań makefile naszego projektu, a także porady oraz przykłady użycia.

all	Wykonuje zadanie *client* oraz *server*. Wymaga pliku *java.policy*.
server	Kompiluje pakiet serwera w Javie. Tworzy nagłówki JNI serwerów dekompozycji oraz rozwiązywania równań. Kompiluje kod C oraz tworzy z niego bibliotekę współdzieloną z funkcjami natywnymi. Może wykonać zadanie *$(TARGETDIR)*. Wykonuje zadanie *common*.
client	Kompiluje pakiet klienta w Javie. Może wykonać zadanie *$(TARGETDIR)*. Wykonuje zadanie *common*.
$(TARGETDIR)	Tworzy strukturę katalogów dla plików wyjściowych.
clean	Usuwa pliki nagłówkowe JNI oraz ścieżkę docelową kompilacji wraz z zawartością.
runclient	Uruchamia klienta. Domyślne parametry: *DHOST=taurus* – nazwa hosta z serwerem do dekompozycji LU; *DPORT=1444* – numer portu, na którym nasłuchuje serwer do dekompozycji *SHOST=taurus* – nazwa hosta z serwerem do rozwiązywania układu równań; *SPORT=1445* – numer portu, na którym nasłuchuje serwer do rozwiązywania układu równań; *AFILE=resources/A3.txt* – lokalizacja pliku z macierzą A; *BFILE=resources/b3.txt* – lokalizacja pliku z wektorem b; *XFILE=resources/x3.txt* – ścieżka, pod którą zostanie zapisany plik z wektorem rozwiązań; **WAŻNE!** Przed uruchomieniem tego zadania proszę wykonać komendę *make all*, *make rundserver* oraz *make runsserver*.

rundserver	Uruchamia serwer dekompozycji. Domyślny parametr: *DPORT=1444* – port, na którym będzie nasłuchiwał serwer; **WAŻNE!** Przed uruchomieniem tego zadania proszę wykonać komendę *make server*.
runsserver	Uruchamia serwer rozwiązywania układu równań. Domyślny parametr: *SPORT=1445* – port, na którym będzie nasłuchiwał serwer; **WAŻNE!** Przed uruchomieniem tego zadania proszę wykonać komendę *make server*.

7.1. Porady

1. Sprawdź, czy parametr *JAVA_HOME* znajdujący się w pliku *Makefile* zawiera poprawną ścieżkę do katalogu głównego Javy.
   Domyślnie: *JAVA_HOME= /usr/lib/jvm/java-8-oracle*