Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych
Projekt JA

Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot:	Grupa:	Sekcja:
2016/2017	**SSI**	**Języki Asemblerowe**	1	1

Imię:	**Dominik**	**Prowadzący**: OA/BZ/GD/HM/JP	JP
Nazwisko:	**Rączka**		

Raport końcowy

Temat:

Program do rozwiązywania układów równań liniowych metodą Choleskiego

Data: dd/mm/yyyy	13/02/2017

1. Temat projektu i opis założeń

Celem projektu było napisanie aplikacji zawierającej algorytm, który będzie wykonany w bibliotece języka wysokiego poziomu oraz w bibliotece języka asembler.
Mój program rozwiązuje układy równań liniowych z wykorzystaniem metody Choleskiego. Główna część programu, która obsługuje GUI oraz interakcję z użytkownikiem została napisana w języku C# z wykorzystaniem WPF oraz biblioteki wysokiego poziomu w języku C i biblioteki niskiego poziomu w asemblerze.

2. Analiza zadania

Metoda Choleskiego jest jedną z metod rozwiązywania układów równań liniowych. Polega ona na odpowiednim podzieleniu macierzy A (macierz w której zapisujemy równania) na macierze pomocnicze L oraz U przy pomocy metody eliminacji Gaussa, a następnie wyliczeniu wektora wynikowego. Jest to sposób, który pozwala szybko i dokładnie obliczać macierze o dużych rozmiarach komputerom, jednak dla człowieka jest on nieefektywny, ponieważ wymaga wykonania wielu operacji matematycznych.
Wzory opisujące metodę obliczeń, zakładając układ opisany równaniem **AX=B:**

$$A=LU$$

$$\text{gdzie} \quad L=\begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \qquad U=\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}.$$

Macierze L i U liczymy według poniższych wzorów:

$$l_{ik}=\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad k=1,2,\ldots,n \quad i=k+1,\ldots,n.$$

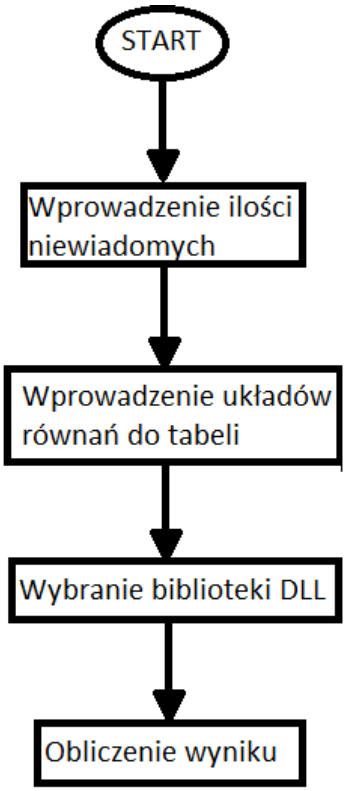
$$u_{ki}=a_{ki}^{(k)} \qquad k=1,2,\ldots,n \quad i=k,k+1,\ldots,n.$$

Algorytm rozwiązania układu równań AX=B jest następujący – rozbijając równanie LUX=B otrzymujemy dwa równania LY=B i UX=Y. Z pierwszego równania wyznaczamy wektor Y a z drugiego wektor X, według poniższych wzorów:

$$\begin{cases} y_1=b_1 \\ y_i=b_i-\sum_{k=1}^{i-1} l_{ik}y_k \quad i=2,3,\ldots,n \end{cases}$$

$$\begin{cases} x_n=\dfrac{y_n}{u_{nn}} \\ x_i=\dfrac{y_i-\sum_{k=i+1}^{n} u_{ik}x_k}{u_{ii}} \quad i=n-1,n-2,\ldots,1 \end{cases}$$

3. Schemat blokowy programu



Rysunek 1. Schemat blokowy głównego programu.

4. Program główny i zmienne globalne

Interfejs graficzny programu oraz obsługa bibliotek została napisana w języku C# z użyciem WPF. Interfejs użytkownika pozwala nam wprowadzić macierz do rozwiązania oraz wyświetla informacje o wyniku działań matematycznych i czasie działania bibliotek. Ponadto sprawdza czy dane wpisane przez użytkownika są poprawne.

Poniżej przedstawiam nagłówki najważniejszych funkcji programu głównego wraz z ich krótkim opisem:

```
private void matrixCreation()
	Metoda tworząca macierze i wektory wykorzystywane do obliczeń.
private bool createLU()
	Metoda zapełniająca macierze L i U odpowiednimi współczynnikami oraz sprawdzająca
poprawność podanych danych (zapobiegnięcie dzieleniu przez 0).
private double[] matrixToArray(double[,] matrix)
	Metoda konwertująca macierze zapisane w tablicach dwuwymiarowych na tablice
jednowymiarowe, w celu łatwiejszego przesłania danych do bibliotek.
public double[] calculate()
	Metoda, która wybiera odpowiednią bibliotekę do wykonania obliczeń.
private void validateResult()
	Metoda sprawdzająca czy uzyskany wynik jest liczbą.
private void equationsDatagridCreation()
	Metoda przygotowująca tabelę do wpisania danych wejściowych.
private void resultDatagridCreation(double[] vectorX)
	Metoda tworząca tabelę z wektorem wynikowym.
```

5. Funkcje bibliotek

Obie biblioteki zawierają jedną funkcję - CalculateC dla biblioteki wysokiego poziomu i CalculateASM dla biblioteki niskiego poziomu. Realizuje ona obliczanie wektora Y, a następnie wektora X, czyli wektora, którego współczynniki spełniają zadany przez użytkownika układ równań liniowych. Wzory realizowane przez biblioteki:

$$\begin{cases} y_1 = b_1 \\ y_i = b_i - \sum_{k=1}^{i-1} l_{ik} y_k \quad i = 2, 3, \ldots, n \end{cases}$$

$$\begin{cases} x_n = \dfrac{y_n}{u_{nn}} \\ x_i = \dfrac{y_i - \sum_{k=i+1}^{n} u_{ik} x_k}{u_{ii}} \quad i = n-1, n-2, \ldots, 1 \end{cases}$$

a) Biblioteka w języku C:

```
void calculateC(int size, double* matL, double* matU, double* vecY, double* vecB,
double* vecX)
```

Funkcja jako argumenty przyjmuje wszystkie dane potrzebne do wykonania obliczeń – rozmiar macierzy, oraz wskaźniki na macierze i wektory wykorzystywane podczas obliczeń.

b) Biblioteka w języku asemblera:

```
calculateASM PROC
```

Procedura w języku asemblera pobiera 4 dane poprzez rejestry (rcx – size, rdx -*matL, r8 - *matu, r9 - *vecY), a pozostałe dwie, czyli *vecB oraz *vecX pobiera ze stosu, ponieważ konwencja wywołania w architekturze x64 nie pozwala na przekazanie większej ilości parametrów przez rejestry Są one zapisywane w rejestrach r10 (*vecB) oraz r11 (*vecX). Rejestry również wykorzystywane w procedurze:

- `r12` – jest w nim przetrzymywany iterator zewnętrznych pętli
- `r13` – jest w nim przetrzymywany iterator wewnętrznych pętli
- `r14` – przechowuje 0 do porównań oraz zerowania rejestrów
- `r15` – używany jest do obliczania adresów
- `xmm0` – używany jest do sumowania iloczynów w pętlach
- `xmm1, xmm2` – używane są do wykonywania pozostałych obliczeń matematycznych

Ponieważ obliczenia są przeprowadzane na liczbach zmiennoprzecinkowych typu `double` użycie rozkazów SSE było naturalne i nie wymagało specjalnych modyfikacji w kodzie procedury.

6. Struktura danych wejściowy/testowych

Program przyjmuje jako ilość niewiadomych równania liczbę naturalną z zakresu <1, 17>. Górne ograniczenie tego przedziału jest podyktowane komfortem użytkowania programu – większa liczba niewiadomych mogłaby znacząco zmniejszyć przejrzystość interfejsu oraz doprowadzić do przepełnienia pamięci RAM. Do poszczególnych pól tabeli można wprowadzać liczby zmiennoprzecinkowe z zakresu 1.7e +/-308 (dokładność 15 cyfr po przecinku). Należy również pamiętać, aby jako separatora dziesiętnego używać znaku '.' zamiast znaku ',' – w innym wypadku dane będą źle zinterpretowane przez aplikację.

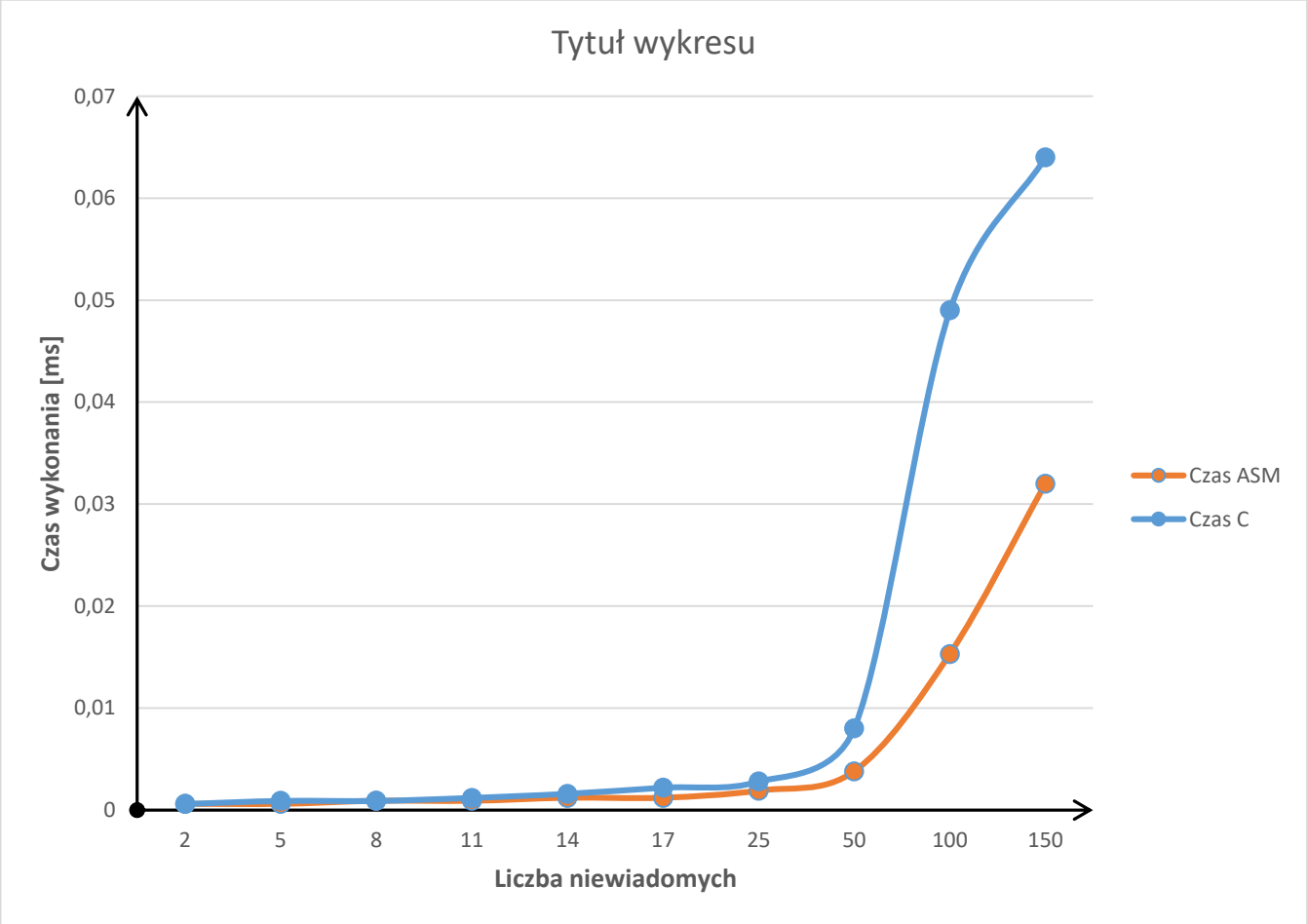
7. Uruchamianie i testowanie

Program uruchamia się z pliku wykonywalnego .exe. Program jest zabezpieczony przed wprowadzeniem błędnych danych (liter zamiast cyfr lub liczb spoza obsługiwanych przedziałów) lub niewybraniem wszystkich opcji niezbędnych do uruchomienia obliczeń. Wprowadzając dane do obliczeń zawierające separator dziesiętny należy pamiętać o zastosowaniu znaku '.' zamiast ','. W przeciwnym razie program pominie separator i złączy całkowitą część liczby z jej częścią ułamkową ignorując separator dziesiętny w postaci ','. Uruchomienie programu z niepoprawnymi danymi jest niemożliwe – o ewentualnych błędach poinformuje użytkownika odpowiedni komunikat.

Poprawna konfiguracja danych wejściowych programu pozwala uzyskać wynik zadanego przez nas układu równań liniowych oraz poznać czas wykonania funkcji zawartych w bibliotekach.

8. Porównanie czasów wykonania

Początkowo do testów użyłem tylko układów równań liniowych z ilością niewiadomych z obsługiwanego przez program zakresu. Jednak różnice w czasach wykonania funkcji w obu bibliotekach były bardzo małe, a sam czas wykonania obu bibliotek bardzo podobny, dlatego do testów postanowiłem odblokować maksymalną ilość niewiadomych w programie, aby przeprowadzić dokładniejsze testy. Podczas testów nie stwierdziłem wpływu rodzaju danych na czas obliczeń (wielkość i znak liczb). Wyniki testów przedstawiam poniżej na wykresie:

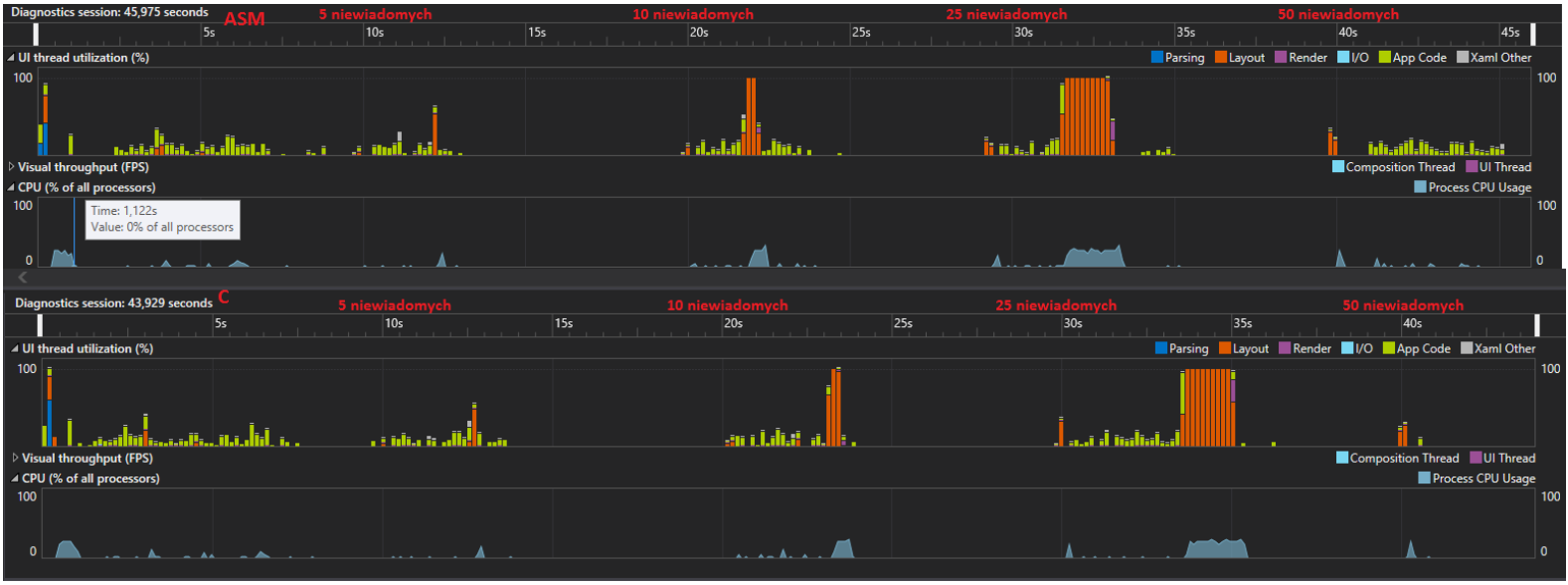


Jak można odczytać z wykresu, czas działania obu bibliotek zaczyna się znacząco różnić dopiero przy ilości niewiadomych powyżej 25, czyli poza zakresem obsługiwanym przez program. Testowanie postanowiłem zakończyć na liczbie 150 niewiadomych, ponieważ pobór pamięci RAM zaczął przekraczać 500MB i generowanie tablicy z danymi testowymi zaczęło trwać znacząco dłużej niż obliczanie wyników (generowanie tablicy z danymi trwało około 15s.). Wykres czasu działania funkcji rośnie wykładniczo, tak samo jak ilość danych do przeliczenia.

Miłym zaskoczeniem dla mnie było to, że udało mi się napisać w języku asemblera kod, który działa szybciej niż jego odpowiednik w języku C przy zastosowaniu standardowej optymalizacji. Niestety po włączeniu optymalizacji nastawionej na uzyskanie maksymalnej szybkości działania czas wykonania kodu w C znacząco spadł i stał się szybszy od mojego kodu w asemblerze. Czas wykonania po włączeniu optymalizacji, dla układu równań ze 150-cioma niewiadomymi wynosił ok. 0,035ms w asemblerze oraz 0,025ms w języku C.

9. Analiza działania z wykorzystaniem modułu profilera VS2015

Do analizy z wykorzystaniem profilera użyłem układów równań 5-cioma, 10-cioma, 25-cioma i 50-cioma niewiadomymi. Zrezygnowałem z testowania programu dla większej ilości niewiadomych gdyż profiler miał później problem z wygenerowaniem raportu diagnostycznego. Było to najprawdopodobniej spowodowane dużym obciążeniem procesora podczas generowania danych testowych. Poniżej zamieszczam raporty diagnostyczne wygenerowane podczas testów:

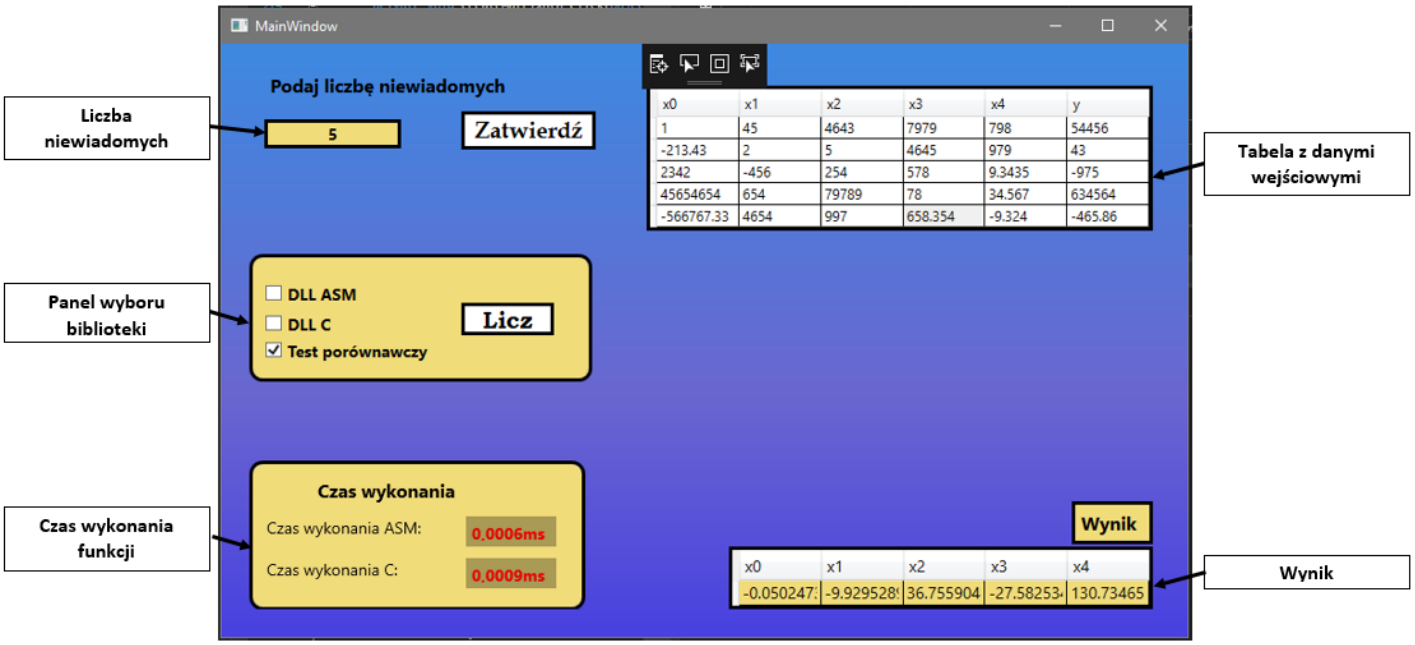


Jak można odczytać z wykresów wygenerowanych przez profiler, użycie procesora podczas samych obliczeń (10, 20, 30 i 40 sekunda) nie było duże. Znacznie większym obciążeniem było wygenerowanie tabeli z danymi testowymi. Dla tych danych testowych różnice między użyciem biblioteki w asemblerze i w języku C są minimalne.

10. Instrukcja obsługi

Aby uruchomić program należy podać liczbę niewiadomych układu równań, a następnie nacisnąć przycisk „Zatwierdź”. Do wygenerowanej tabeli należy wprowadzić poszczególne współczynniki zgodnie z oznaczeniami kolumn. Następnie wystarczy wybrać interesująca nas bibliotekę i nacisnąć przycisk „Licz”. Pod tabelą z danymi wejściowymi pojawi się tabela z wektorem będącym wynikiem wprowadzonego układu równań liniowych, a w lewym dolnym rogu pojawi się informacja o czasie wykonania funkcji w wybranej przez nas bibliotece.

Opis okna aplikacji:



11. Wnioski

Podczas realizacji tego projektu mogłem lepiej zapoznać się z programowaniem w języku asemblera oraz instrukcjami SSE.

Wydajność biblioteki napisanej przeze mnie w języku asemblera była niestety niższa od wydajności biblioteki w języku C zoptymalizowanej przy użyciu środowiska Visual Studio. Ponadto implementacja algorytmu z języku asemblera zajęła mi więcej czasu, gdyż musiałem dużo uwagi poświęcić odpowiedniemu operowaniu na adresach pamięci.

Przez takie rezultaty uważam, że w obecnych czasach programowanie w języku asemblera staje się nieefektywne, gdyż wymaga większych nakładów pracy w zamian nie dając gwarancji uzyskania lepszych wyników. Być może podczas pisania skomplikowanego algorytmu przez osobę, która bardzo dobrze zna język asemblera i jego możliwości wyniki byłyby inne.

12. Literatura

x86.renejeschke.de/
docs.oracle.com/cd/E26502_01/html/E28388/enmzx.html#scrolltoc
software.intel.com/en-us/
msdn.microsoft.com/en-us/