Instytut Informatyki Politechniki Śląskiej

Zespół Mikroinformatyki i Teorii Automatów Cyfrowych

Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/SMiW):	Grupa	Sekcja
2016/2017	**SSI**	**Języki Asemblerowe**		**8**

Imię:	**Kamil**	**Prowadzący**: OA/JP/KT/GD/BSz/GB	**OA**
Nazwisko:	**Ziętek**		

Raport końcowy

Temat projektu:

Rozwiązywanie układów równań liniowych metodą Choleskiego (macierzy LU)

Data oddania: dd/mm/rrrr	**05/02/2017**

1. **Temat projektu i opis założeń**

Tematem projektu jest utworzenie aplikacji okienkowej rozwiązującej układy równań liniowych metodą Choleskiego (zwaną również metodą macierzy LU). W głównym oknie aplikacja umożliwia wczytanie danych z pliku, podgląd wczytanych danych oraz wyników, a także wybór biblioteki z pomocą której będą wykonywane obliczenia (assembler lub C). Dane wczytywane są z pliku tekstowego, zawierającego kolejne współczynniki układu równań liniowych. Po wykonaniu obliczeń pokazywany jest czas ich trwania.

Program składa się z trzech modułów: MAIN, DLL_ASM oraz DLL_C. Główny moduł to program w języku C#, stworzony z wykorzystaniem Windows Forms, odpowiadający za interfejs graficzny oraz wczytywanie danych z pliku tekstowego do pamięci. Biblioteka DLL w asemblerze oraz w C odpowiada za część obliczeniową.

2. Analiza zadania

Rozwiązanie układu równań liniowych zapisanego w postaci macierzy współczynników A oraz wektora B, polega na znalezieniu wartości zmiennych x (wektora X). Ogólny zapis takiego układu równań w postaci macierzowej to AX = B. Metody rozwiązywania takich układów można podzielić na dwie grupy: metody dokładne (wzory Cramera, metoda Gaussa, metoda Gaussa-Jordana, metoda wyboru elementu podstawowego, metoda Choleskiego czyli rozkładu na macierze L i U) oraz metody iteracyjne (metoda Jacobiego zwana też metodą iteracji prostej, metoda Seidela, metoda relaksacji). Jako projekt wybrałem zaimplementowanie metody z pierwszej grupy, konkretnie metody Choleskiego.

Metoda ta polega na przekształceniu macierzy A w dwie macierze trójkątne L i U, odpowiednio dolną z jedynkami na przekątnej głównej i górną, taką że A = LU. Wstawiając to równanie do ogólnego zapisu postaci macierzowej otrzymujemy równanie LUX = B, które można rozbić na dwa równania:
LY = B oraz UX = Y. Z pierwszego równania wyznaczamy wektor Y, za pomocą którego z drugiego równania wyznaczamy wektor X, będący rozwiązaniem naszego układu.

Rozwiązanie rozpoczynamy od wyznaczenia macierzy U, która jest równa macierzy trójkątnej uzyskanej w eliminacji Gaussa, czyli:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)} a_{kj}^{(k)}}{a_{kk}^{(k)}}$$

(k = 1,2,…,n-1; i,j=k+1,k+2,…,n)

Drugim krokiem jest wyznaczenie macierzy L, której elementy kolejnych kolumn są równe współczynnikom, przez które mnożone są w kolejnych krokach wierze układu równań celem dokonania eliminacji niewiadomych w odpowiednich kolumnach:

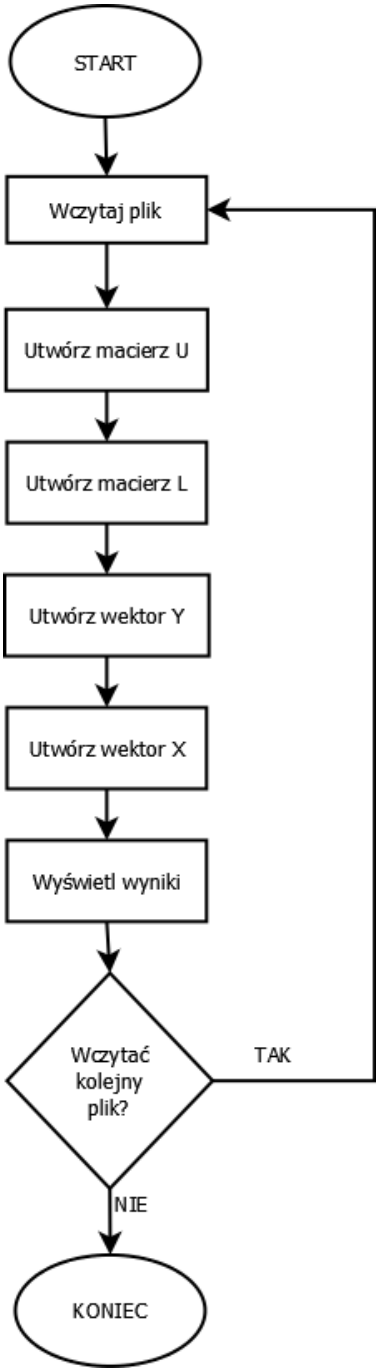
$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad k = 1, 2, \ldots, n \quad i = k+1, \ldots, n$$

Następnie należy wyznaczyć wektor Y, a później na jego podstawie wektor X, będący rozwiązaniem zadania.

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

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

3. Schemat blokowy programu



4. Opis programu w języku wysokiego poziomu

W języku C# zaimplementowany został graficzny interfejs całej aplikacji, a także wczytywanie danych z pliku tekstowego. Można w nim wybrać bibliotekę za pomocą której będą wykonywane obliczenia, oraz sprawdzić czas wykonywania. Program składa się z następujących plików źródłowych:
- Program.cs, będący punktem startowym aplikacji
- Form1.cs, będące formatką zawierającą główne okno aplikacji o obsługę interfejsu
- ExternalFunctions.cs, będące klasą przekazującą parametry do zewnętrznych bibliotek oraz wywołującą zawarte w nich funkcje
- vievMatrix.cs, będące podglądem wczytanej macierzy

Zmienne globalne:
- bool assembly, będąca oznaczeniem wybranej przez użytkownika biblioteki (przyjmuje wartość true jeśli wybrana jest biblioteka asemblerowa, false jeśli biblioteka C)
- bool computed, wskazująca czy macierz została tylko wczytana do program (false), czy też obliczenia zostały już na niej wykonane (true)
- int size, przechowująca wymiar macierzy A, a więc jednocześnie wymiar wszystkich pozostałych macierzy oraz długość wszystkich wektorów, potrzebne przy przekazywaniu ich do funkcji zewnętrznych (przekazujemy wówczas tylko adres pierwszego elementu tablicy)
- double[][] matrixA, będąca wczytaną z pliku macierzą
- double[] vectorB, będący wczytanym z pliku wektorem wyrazów wolnych
- double[] vectorX, będący wynikiem obliczeń – na starcie programu jest on pusty, dopiero w wyniku działania funkcji bibliotecznych umieszczane w nim są wyniki obliczeń

5. Opis funkcji bibliotek DLL

Funkcje biblioteczne są 4 w każdej wersji biblioteki. Służą one do wykonywania kolejnych kroków algorytmu wyznaczania wyników układu równań – po kolei jest to wyliczenie macierzy U, wyliczenie macierzy L, wyliczenie wektora Y oraz wyliczenie wektora X. Każda funkcja ma nazwę odpowiadającą wykonywanemu przez nią działaniu, dodatkowo zawierając po znaku podkreślenia „_” oznaczenie, czy pochodzi ona z biblioteki asemblerowej (_ASM), czy C (_C). Parametry w przypadku metod w obu odpowiadających sobie bibliotekach są takie same. Każda z metod przyjmuje w parametrach macierze „spłaszczone” do jednowymiarowej tablicy, oraz rozmiar tych macierzy, wymagane do obliczenia danej wartości.
- computeMatrixU: matrixU oraz size,
- computeMatrixL: matrixL, matrixU oraz size,
- computeVectorY: vectorY, vectorB, matrixL oraz size,
- computeVectorX: vector, vectorY, matrixU oraz size.

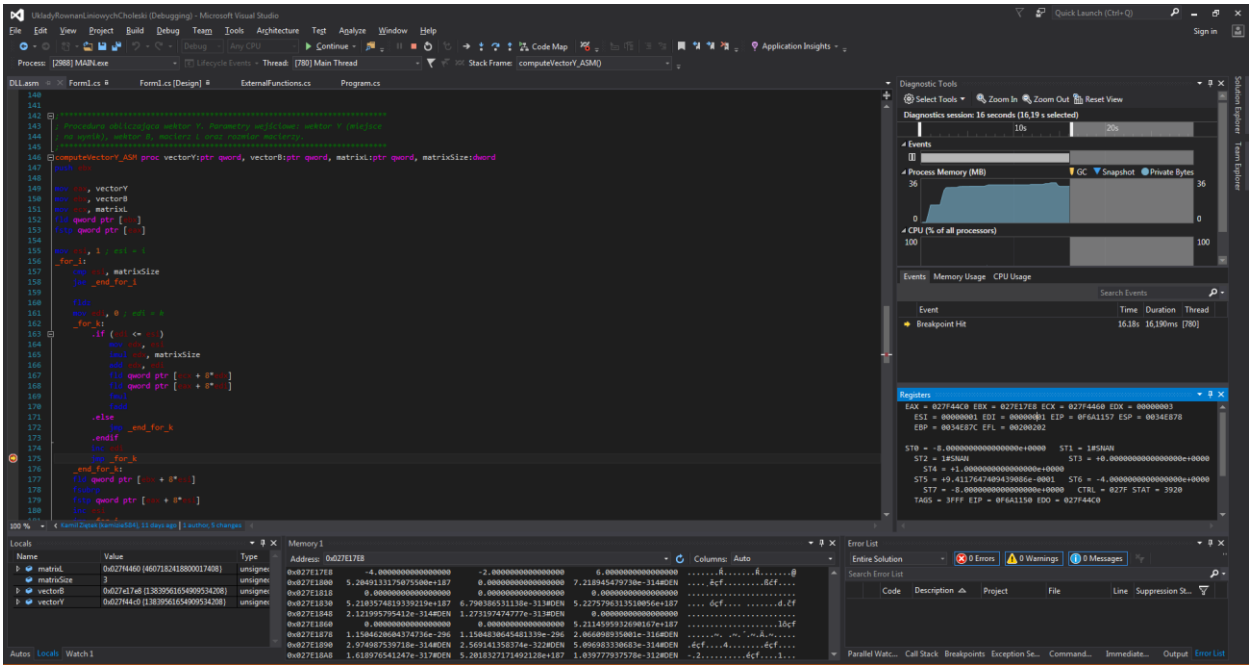
6. Opis struktury danych wejściowych/testowych programu

Program odczytuje jedynie pliki tekstowe zawierające poprawnie zapisany układ równań liniowych. W pliku powinna znajdować się najpierw macierz A, następnie wektor B, z kolejnymi wartościami oddzielonymi znakami spacji. Wartości muszą być podawane wierszami (najpierw wszystkie liczby z pierwszego wiersza macierzy A i współczynnik z wektora B, później po znaku nowej linii z drugiego wiersza itd.). Jako że macierz obrazująca układ równań liniowych jest z definicji kwadratowa, dla macierzy o rozmiarze n plik tekstowy musi mieć x wierszy, oraz x+1 kolumn, gdzie w x+1-wszej kolumnie znajdować się będą wartości z wektora B. Przykładowe poprawnie zapisane równania liniowe znajdują się w folderze Dane testowe, z nazwami odpowiadającymi rozmiarom macierzy.

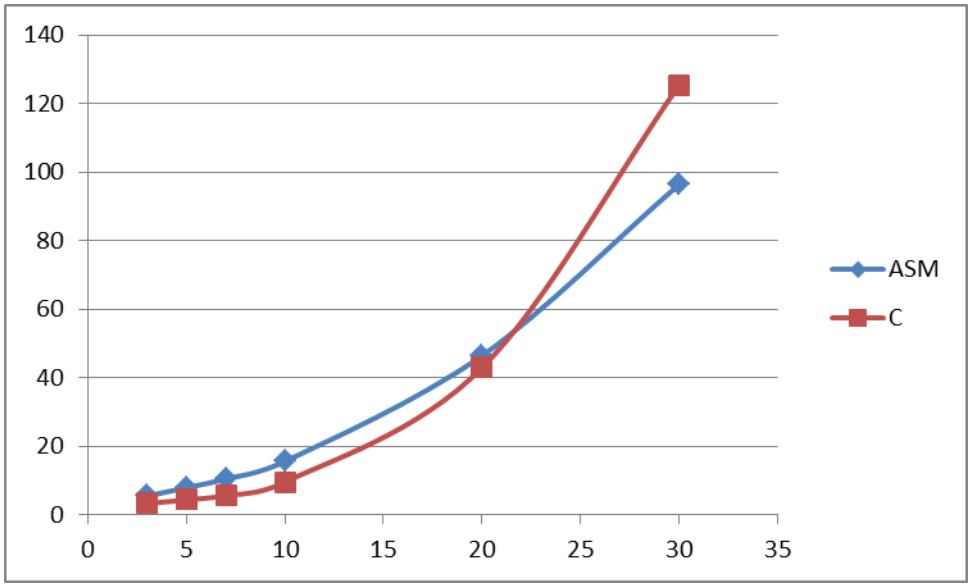
7. Opis uruchamiania programu i jego testowania

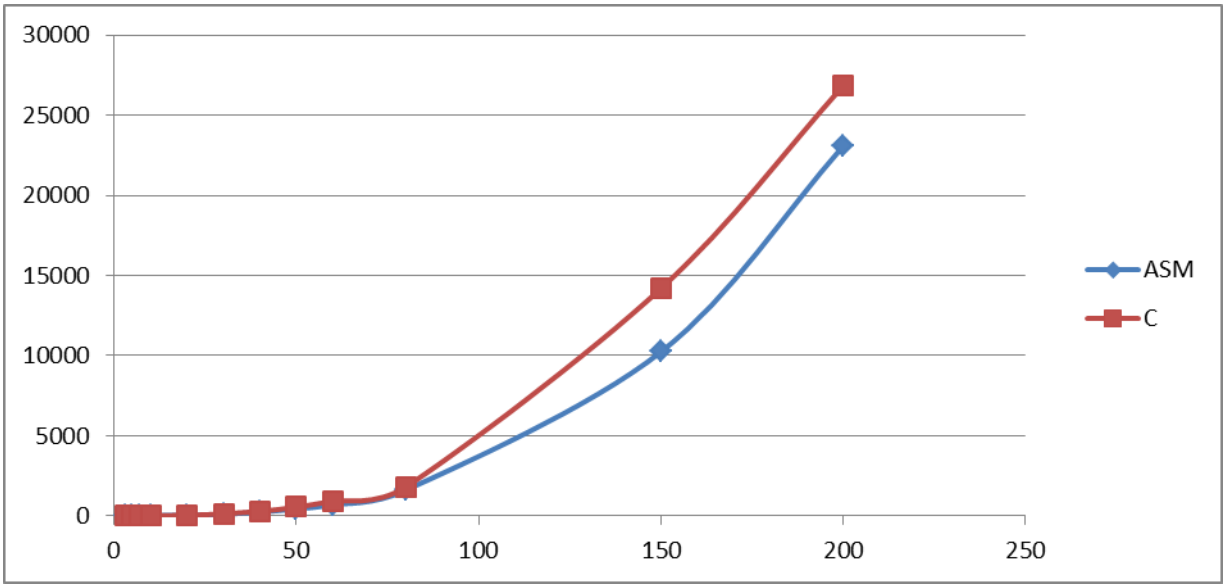
Testowanie programu polegało na wczytywaniu kolejnych układów równań do rozwiązania, oraz kontrolnemu sprawdzaniu poprawności ich wyników. W przypadku mniejszych rozmiarów obliczenia sprawdzające zostały wykonane za pomocą kalkulatora i kartki papieru, w przypadku większych – wyniki zostały porównane z wynikami uzyskanymi w kalkulatorze online, przyjmującym dane w postaci odpowiadającej strukturze plików tekstowych wczytywanych do programu (http://calcoolator.pl/). Po sprawdzeniu poprawności obliczeń dla macierzy o rozmiarze mniejszym lub równym 10, założyłem poprawność działania programu również dla większych danych – algorytm obliczeń jest taki sam, a duża ilość liczb utrudnia ich proste porównywanie.

O ile stworzenie kodu w języku C było stosunkowo proste (wystarczyło odpowiednio zinterpretować algorytm zapisany za pomocą symboli matematycznych, w szczególności uwzględniając sumowanie), a analiza każdego kroku operacji wygodna i intuicyjna (tablice wyświetlane bezpośrednio w podglądzie zmiennych), o tyle debugowanie asemblera było nieco bardziej skomplikowane. Bardzo pomocny okazał się podgląd pamięci i rejestrów, w szczególności możliwość wyświetlania zawartości pamięci pod postacią 64-bitowych liczb zmiennoprzecinkowych oraz podglądu zawartości stosu koprocesora odpowiadającego za operacje zmiennoprzecinkowe. Dzięki temu, możliwe było porównywanie wyników z tymi uzyskanymi za pomocą kodu w C. W celu wygodniejszego tworzenia kodu zainstalowałem również rozszerzenie AsmDude, dodające kolorowanie składni domyślnie niedostępne dla edytora asemblerowego.



Podstawowym testem sprawdzającym wydajność programu jest pomiar czasu wykonywania obliczeń dla różnych danych testowych. Wyniki testów dla zestawu danych umieszczonego w katalogu Dane testowe widoczne są na poniższym wykresie (na osi X rozmiar macierzy, na osi Y czas obliczeń, jednostka – mikrosekundy):

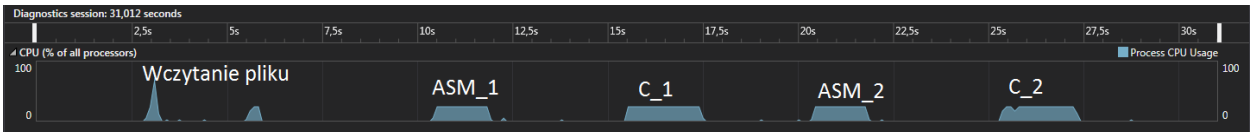




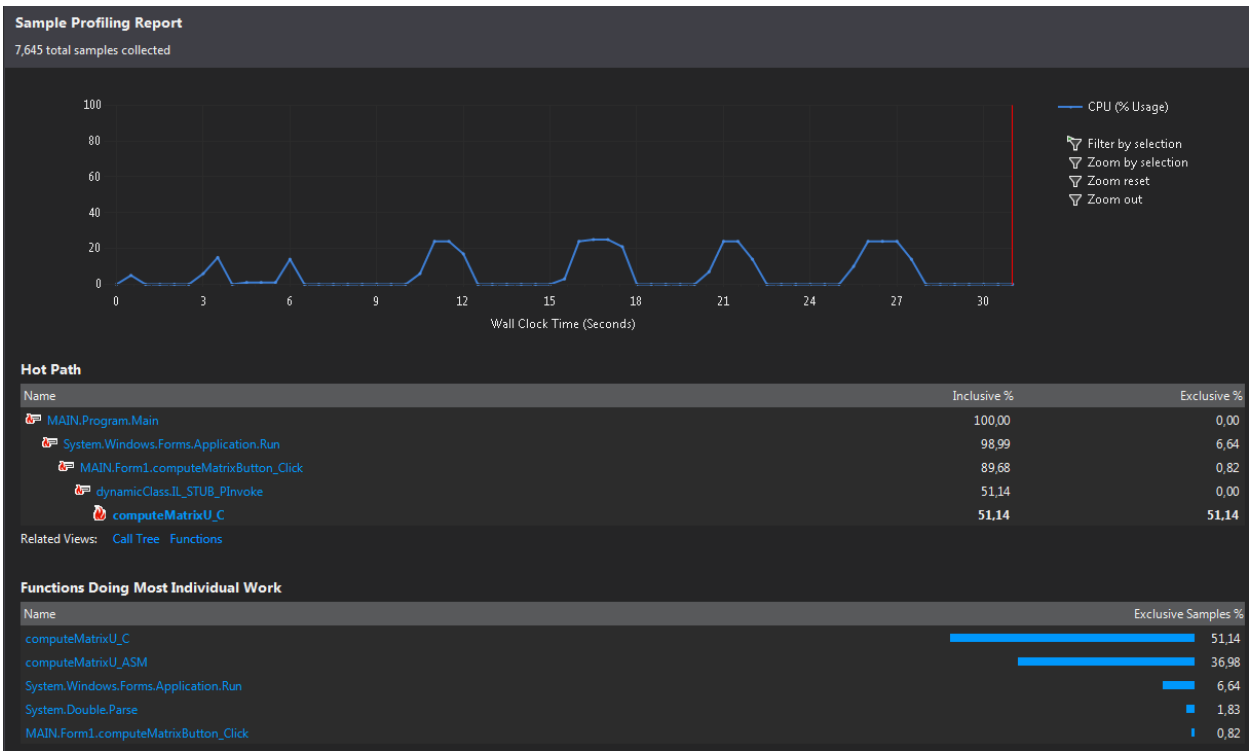
Czas jest średnim czasem wykonania, wyznaczonym na podstawie kilku pomiarów. Jak widać, wyniki są bardzo zbliżone. Dla małych rozmiarów macierzy nieznacznie szybciej wykonuje się kod w C, aby wraz ze wzrostem ilości danych ustąpić miejsca asemblerowi. Potwierdziło to testowe wczytanie do programu macierzy o absurdalnie dużym rozmiarze – tam również czas wykonywania obliczeń w asemblerze był nieco krótszy. Należy jednak zaznaczyć, że wyniki te obarczone mogą być błędem: czas był mierzony za pomocą obiektu typu Stopwatch, który wyznacza jedynie czas jaki upłynął od kliknięcia przycisku do otrzymania wyniku – w tym czasie system operacyjny działa normalnie, więc ostateczny wynik może zależeć również od takich czynników jak zużycie procesora przez inne programy.

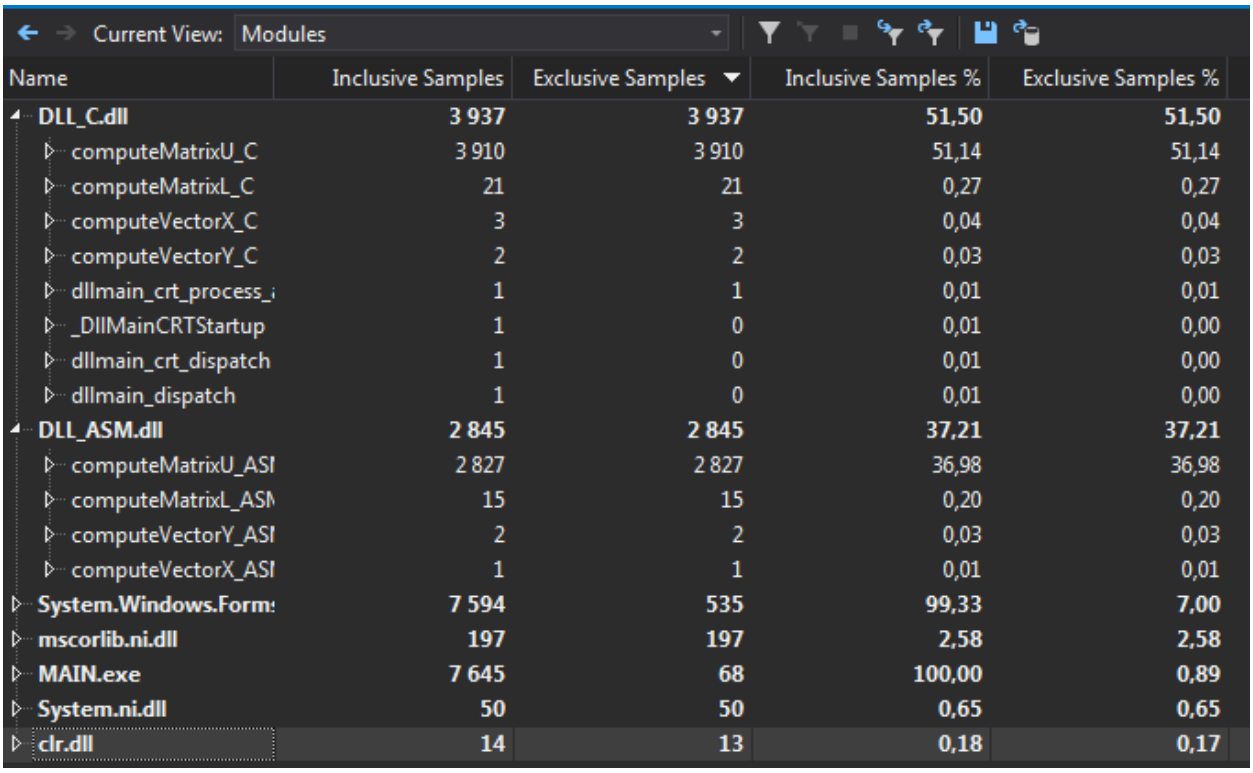
Na wykresach pominięty został również pierwszy czas uzyskany po uruchomieniu programu, zawsze większy od wszystkich następnych. Wydłużenie tego czasu spowodowane jest faktem odczytu biblioteki z dysku: samo uruchomienie okienka nie wczytuje od razu bibliotek do pamięci, są one dopiero ładowane przy pierwszym wykonaniu obliczenia (można to sprawdzić uruchamiając sam plik .exe bez plików .lib w katalogu programu).

Drugim przeprowadzonym testem była analiza zużycia procesora i pamięci, przeprowadzona dostępnym w Visual Studio 2015 profilerem. Tutaj obiektem testowym była macierz o rozmiarze 500, aby uwidocznić różnice. Wykres czasu użycia procesora:

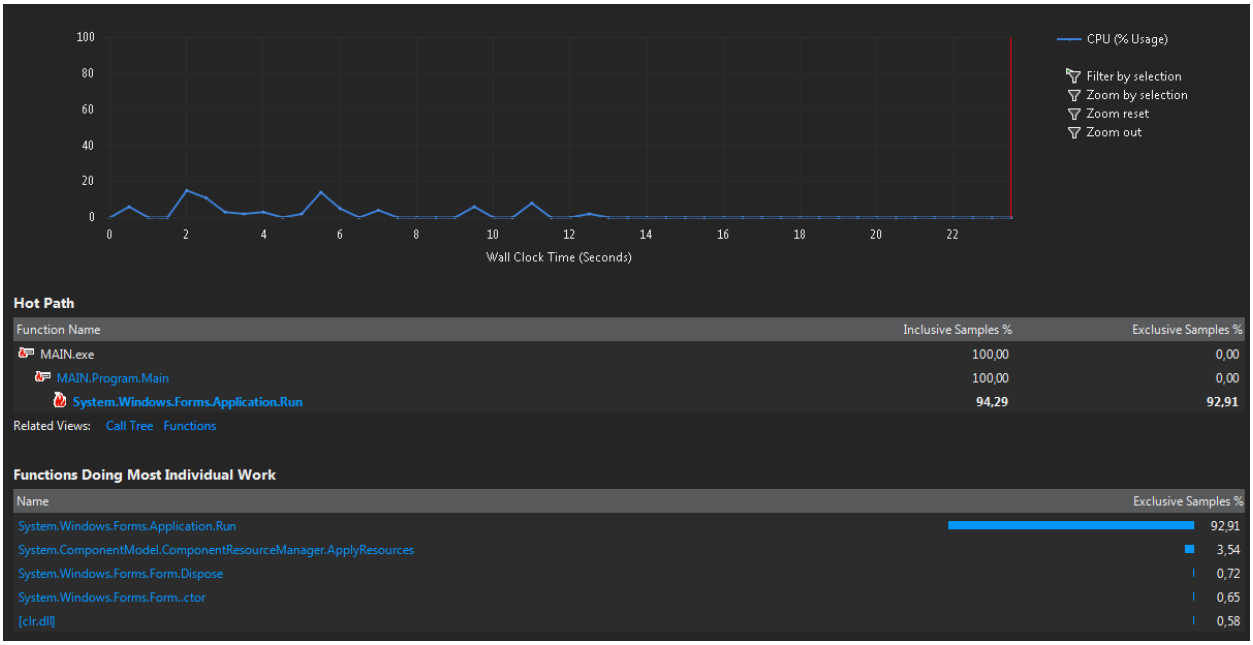


Działanie programu polegało na wczytaniu pliku, wykonaniu obliczeń w asemblerze, w C, a później drugi raz w asemblerze i drugi raz w C. Rezultaty są zbliżone do siebie, jednak sprawiają wrażenie, że kod w Asemblerze wykonuje się szybciej. Potwierdza to podsumowanie, wyszczególniające również poszczególne funkcje z procentowym wykorzystaniem procesora:

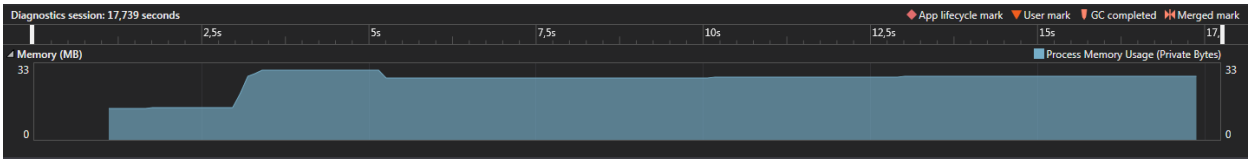




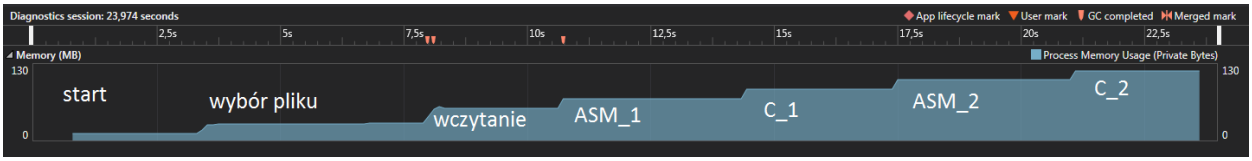
Jak widać, najbardziej wykorzystuje procesor funkcja odpowiadająca za wyliczenie macierzy U (to zrozumiałe, ponieważ wykonywane w niej obliczenia są bardziej skomplikowane od pozostałych). Widać również większe użycie procesora w przypadku wykonywania tych obliczeń w C. Przeprowadzenie analogicznego testu dla mniejszego pliku nie przyniosło oczekiwanego rezultatu: obliczanie trwało tak krótko w porównaniu do chociażby obsługi GUI, że profiler nie zarejestrował wyników dla poszczególnych bibliotek.



Analogiczne testy zostały przeprowadzone dla zużycia pamięci. Tutaj ponownie, dla małych plików zużycie pamięci jest niezauważalne, widać jedynie niewielki wzrost przy otwieraniu pliku:



Sprawa ma się inaczej dla dużego pliku:



Tutaj widać wyraźny wzrost zużycia pamięci po wykonaniu każdej akcji. Wynik również nie jest zaskoczeniem, ani w kwestii ciągłego wzrostu, ani w kwestii takiego samego użycia pamięci dla obu bibliotek. Zarządzanie pamięcią odbywa się po stronie wysokopoziomowego programu w C#, co powoduje zarówno brak różnic między asemblerem i C (biblioteki dostają tylko adres już przydzielonej pamięci, która jest przydzielana w C# w takiej samej ilości dla obu bibliotek), a także ciągły wzrost zużycia pamięci, nawet przy wykonywaniu tych samych obliczeń (miejsce na wyniki jest przydzielane przy wczytywaniu pliku, a jego ewentualnym zwalnianiem zarządza Garbage Collector, wyrzucając niewykorzystywane dane dopiero gdy zabraknie miejsca na zapis nowych).

8. Wnioski

Fakt lepszego działania programu dla dużych plików początkowo trochę mnie zaskoczył – spodziewałem się raczej, że z powodu trudniejszego operowania współczynnikami tablic wynik będzie odwrotny (zamiast napisać np. [i*k+j] trzeba operować bezpośrednio na rejestrach). W rzeczywistości jednak te obliczenia i tak są wykonywane w taki sam sposób, różnią się jedynie zapisem wysokopoziomowym, który po skompilowaniu prowadzi do takiego samego kodu maszynowego. Fakt, że największy wpływ na wydajność ma funkcja wykonująca najwięcej obliczeń zmiennoprzecinkowych każe sądzić, że sedno różnic między bibliotekami leży właśnie w obsłudze koprocesora do operacji na liczbach typu double. Może to mieć znaczący wpływ w przypadku operacji na dużej ilości liczb, jednak w przypadku układów równań liniowych (gdzie raczej nieczęsto rozwiązuje się układu dla 500 niewiadomych) większą wagę ma moim zdaniem czas tworzenia programu, który jest zdecydowanie krótszy w przypadku języków wysokiego poziomu.

9. Literatura

Opis algorytmu został zaczerpnięty z materiałów (książek oraz notatek z wykładów i ćwiczeń) z przedmiotu Metody Statystyczne, z którego zajęcia odbywają się na semestrze trzecim i czwartym.