STUDIA INFORMATICA 2004
Volume 25 Number 1 (57)

Beata BYLINA, Jarosław BYLINA
Uniwersytet Marii Curie-Skłodowskiej w Lublinie, Zakład Informatyki

ROZWIĄZYWANIE UKŁADÓW RÓWNAŃ METODĄ ROZKŁADU WZ Z WYKORZYSTANIEM BIBLIOTEK *BLAS1* I *BLAS2*

Streszczenie. W artykule zaprezentowano strategię wyboru elementu podstawowego, efektywny blokowo-punktowy algorytm rozwiązywania układów równań liniowych metodą rozkładu WZ macierzy na czynniki z wyborem elementu podstawowego oraz szczegóły jego implementacji z wykorzystaniem bibliotek *BLAS*. Przedstawiony algorytm rozkładu WZ jest szybszy niż klasyczny rozkład sekwencyjny – również niż ten z włączoną optymalizacją kompilatora.

Słowa kluczowe: rozkład WZ, algorytmy blokowe, *BLAS*, wybór elementu podstawowego

SOLVING LINEAR SYSTEMS USING THE METHOD OF WZ FACTORIZATION WITH *BLAS1* AND *BLAS2* LIBRARIES

Summary. In this article we want to present a strategy of pivoting, an efficient matrix-vector algorithm for solving linear systems by WZ matrix factorization with pivoting and details of its implementation with usage of *BLAS* libraries. The presented algorithm is faster than the sequential one.

Keywords: WZ factorization, block algorithms, *BLAS*, pivoting

1. Wprowadzenie

Rozwiązywanie układów równań

$$\mathbf{Ax} = \mathbf{b}, \text{ gdzie } \mathbf{A} \in \Re^{n \times n}, \mathbf{b} \in \Re^{n} \tag{1}$$

jest ważnym problemem w naukowych obliczeniach inżynierskich (takich jak np. modelowanie). Dla dużych układów równań ważny jest czas obliczeniowy, a on zależy (między innymi) od efektywności poszczególnych podprogramów. Aby program był wykonywany

w krótszym czasie, można kompilować go przy użyciu dostępnych opcji kompilatora bądź też można wykorzystywać wysoko wydajne biblioteki. W niniejszym artykule przedstawiono oba podejścia do rozwiązywania układów równań liniowych metodą rozkładu WZ – z wykorzystaniem wysoko wydajnych bibliotek *BLAS* oraz z włączoną opcją optymalizacyjną kompilatora (`gcc -O3`).

Jedną z metod rozwiązywania gęstych układów równań liniowych postaci (1) jest właśnie rozkład WZ na czynniki. Macierz A jest rozłożona do postaci: **A=WZ**, gdzie macierze **W** i **Z** oraz sposób rozwiązywania układów równań opisane są w sekcji 2. Taki rozkład istnieje dla macierzy nieosobliwej, co zostało pokazane w [2]. W sekcji 3 przedstawiono za [5] i [7] ideę algorytmu rozwiązywania równań (1) metodą rozkładu WZ macierzy na czynniki oraz strategię wyboru elementu podstawowego. W sekcji 4 opisano efektywny algorytm rozkładu WZ macierzy na czynniki z wyborem elementu podstawowego, używając do tego notacji macierzowo-wektorowej.

Następnie w sekcji 5 zaprezentowano szczegóły implementacji oraz wyniki eksperymentu numerycznego, jaki przeprowadzono dla algorytmów: sekwencyjnego rozkładu WZ oraz blokowo-punktowego rozkładu WZ. W sekcji 6 podsumowano wyniki eksperymentów.

2. Rozkład WZ macierzy na czynniki

Sekcja krótko przedstawia metodę rozkładu WZ rozwiązywania układów równań (1). Niech A będzie kwadratową macierzą nieosobliwą. Rozkład WZ (opisany w [2]) **A=WZ**, w którym macierze **W** i **Z** mają następujące kolumny wektorów $\mathbf{w}_i$ i wektory wierszy $\mathbf{z}_i$:

$$
\begin{aligned}
\mathbf{w}_i &= (\underbrace{0,\ldots,0}_{i},1,w_{i+1,i},\ldots,w_{n-i,i},0,\ldots,0)^T, i = 1,\ldots,m,\\
\mathbf{w}_i &= (\underbrace{0,\ldots,0}_{i},1,0,\ldots,0)^T, i = p,q,\\
\mathbf{w}_i &= (\underbrace{0,\ldots,0}_{n-i+1},w_{n-i+2,i},\ldots,w_{i-1,i},1,0,\ldots,0)^T, i = q+1,\ldots,n,\\
\mathbf{z}_i^T &= (\underbrace{0,\ldots,0}_{i-1},z_{ii},\ldots,z_{i,n-i+1},0,\ldots,0), i = 1,\ldots,p,\\
\mathbf{z}_i^T &= (\underbrace{0,\ldots,0}_{n-1},z_{i,n-i+1},\ldots,z_{ii},0,\ldots,0), i = p+1,\ldots,n,
\end{aligned}
\tag{2}
$$

gdzie

$$m = \lfloor (n-1)/2 \rfloor, \quad p = \lfloor (n+1)/2 \rfloor, \quad q = \lceil (n+1)/2 \rceil.$$

Na przykład dla przypadku, gdy $n = 5$ i $n = 6$, zachodzi:

$$\mathbf{W}=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ w_{21} & 1 & 0 & 0 & w_{25} \\ w_{31} & w_{32} & 1 & w_{34} & w_{35} \\ w_{41} & 0 & 0 & 1 & w_{45} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},\quad \mathbf{Z}=\begin{bmatrix} z_{11} & z_{12} & z_{13} & z_{14} & z_{15} \\ 0 & z_{22} & z_{23} & z_{24} & 0 \\ 0 & 0 & z_{33} & 0 & 0 \\ 0 & z_{42} & z_{43} & z_{44} & 0 \\ z_{51} & z_{52} & z_{53} & z_{54} & z_{55} \end{bmatrix},\quad n=5$$

$$\mathbf{W}=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ w_{21} & 1 & 0 & 0 & 0 & w_{26} \\ w_{31} & w_{32} & 1 & 0 & w_{35} & w_{36} \\ w_{41} & w_{42} & 0 & 1 & w_{45} & w_{46} \\ w_{51} & 0 & 0 & 0 & 1 & w_{56} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},\quad \mathbf{Z}=\begin{bmatrix} z_{11} & z_{12} & z_{13} & z_{14} & z_{15} & z_{16} \\ 0 & z_{22} & z_{23} & z_{24} & z_{25} & 0 \\ 0 & 0 & z_{33} & z_{34} & 0 & 0 \\ 0 & 0 & z_{43} & z_{44} & 0 & 0 \\ 0 & z_{52} & z_{53} & z_{54} & z_{55} & 0 \\ z_{61} & z_{62} & z_{63} & z_{64} & z_{65} & z_{66} \end{bmatrix},\quad n=6$$

Po wykonaniu rozkładu można rozwiązać dwie pary układów, używając macierzy **W** i **Z** do otrzymania rozwiązania **x** (**c** jest tu pośrednim wektorem pomocniczym):

$\mathbf{Wc}=\mathbf{b}$,

$\mathbf{Zx}=\mathbf{c}$.

3. Algorytm sekwencyjny rozkładu WZ

Klasyczny algorytm rozwiązywania układów równań (1) metodą rozkładu WZ składa się z dwóch części: redukcji macierzy **A** do macierzy **Z** oraz wyznaczania rozwiązania równania **x**. Pierwsza część polega na wyzerowaniu kolejnych fragmentów kolumn macierzy **A**. W pierwszym kroku zeruje się elementy w pierwszej i n-tej kolumnie od drugiego do n–1 wiersza. Następnie uaktualnia się macierz **A** i wektor **b**. Bardziej formalnie można to zapisać (za pracami [2] oraz [7]) dla pierwszego kroku:

Obliczyć w_{i1} i w_{in} z układu równań:

$$\begin{cases} a_{11}w_{i1}+a_{n1}w_{in}=-a_{i1} \\ a_{1n}w_{i1}+a_{nn}w_{in}=-a_{in}, \quad \text{dla } i=2,\ldots,n-1. \end{cases} \tag{3}$$

Uaktualnić wewnętrzną podmacierz macierzy **A** o rozmiarze o 2 mniejszym od wyjściowego oraz wewnętrzny podwektor wektora **b** – także o rozmiarze o 2 mniejszym:

$$a_{ij} \leftarrow a_{ij}+w_{i1}a_{1j}+w_{in}a_{nj}, \quad j=2,\ldots,n-1, \quad i=2,\ldots,n-1,$$

$$b_i \leftarrow b_i+w_{i1}b_1+w_{in}b_n, \qquad i=2,\ldots,n-1.$$

Analogicznie przeprowadzone zostają następne kroki dla wewnętrznych kwadratowych podmacierzy (o rozmiarze za każdym razem o dwa mniejszym od poprzedniego rozmiaru), która jest zmieniona podczas poprzedniego kroku oraz z podwektorami wektora **b**. Po m

krokach tego procesu otrzymuje się macierz **Z** postaci (2) w miejsce macierzy **A**, zaś w miejsce wektora **b** –wektor **c** wyliczony z układu **Wc**=**b**.

Drugą częścią metody WZ rozwiązywania układów równań jest wyznaczenie szukanego rozwiązania **x** z równania **Zx**=**b**. Metoda polega na rozwiązaniu pewnego układu dwu równań i wyznaczeniu z nich x_p oraz x_q, a następnie uaktualnieniu wektora **b**. Formalnie można to zapisać dla pierwszego kroku następująco:

1) Obliczyć x_p i x_q z układu:

$$\begin{cases} z_{pp}x_p + z_{pq}x_q = b_p \\ z_{qp}x_p + z_{qq}x_q = b_q \end{cases} \tag{4}$$

Zaktualizować wektor **b**:

$$b_i \leftarrow b_i - z_{ip}x_p - z_{iq}x_q, \quad i = 1,\ldots,p-1,q+1,\ldots,n. \tag{5}$$

Dla n nieparzystego układ (4) redukuje się (w pierwszym kroku) tylko do jednego równania. Analogicznie przeprowadza się następne kroki dla wewnętrznych układów równań 2×2.

3.1. Strategia wyboru elementu podstawowego

Należy zauważyć, że rozkład WZ nie działa, jeśli wyznacznik główny układu (3) jest równy zero, tzn. gdy w k-tym kroku zachodzi równość $a_{kk}a_{n-k+1,n-k+1} = a_{k,n-k+1}a_{n-k+1,k}$. Zaradzić temu można zamieniając kolejność równań, a więc poprzez tzw. wybór częściowy (ang. *pivoting*). Zaproponowano następującą strategię wyboru. W każdym kroku (kroki indeksowane są indeksem k) należy wyznaczyć takie i_1 oraz i_2 ($k \le i_1 < i_2 \le n-k+1$), by:

$$\left|\det\begin{bmatrix} a_{i_1,k} & a_{i_1,n-k+1} \\ a_{i_2,k} & a_{i_2,n-k+1} \end{bmatrix}\right| = \max_{k\le i<j\le n-k+1} \left|\det\begin{bmatrix} a_{ik} & a_{i,n-k+1} \\ a_{jk} & a_{j,n-k+1} \end{bmatrix}\right|,$$

a więc żeby dany wyznacznik był co do modułu największy. Następnie zamienia się k-ty wiersz z wierszem i_1 oraz n–k+1 wiersz z wierszem i_2; analogicznie wymienia się wyrazy: b_k z b_{i_1} oraz b_{n-k+1} z b_{i_2}. Otrzymuje się metodę niezawodną, jeśli chodzi o rozwiązanie dla macierzy nieosobliwej, dodatkowo nie grozi nadmiar przy aktualizacji macierzy **A** oraz wektora **b** (podczas obliczania macierzy **Z** oraz wektora **c**). Na potrzeby niniejszego artykułu oznaczono sekwencyjny algorytm rozwiązywania układów równań (1) z opisanym tu wyborem elementu podstawowego nazwą **AWZ**. Wybór elementu podstawowego zapewnia numeryczną poprawność algorytmu, a więc jego najwyższą jakość ze względu na błędy wytworzone przez algorytm **AWZ** w rachunku numerycznym. Jego złożoność czasowa jest rzędu $O(n^3)$.

4. Algorytm blokowo-punktowy

Sekcja przedstawia blokowo-punktowy algorytm rozkładu WZ na czynniki (za [1]) oraz wykorzystujący go algorytm rozwiązywania układu równań liniowych. Jest to efektywny sekwencyjny algorytm, w którym od nowa pogrupowano i uporządkowano operacje skalarne w operacje macierzowo-wektorowe. Opisany tu jest algorytm z wyborem elementu podstawowego, czyli działający dla każdej macierzy nieosobliwej **A**. Strategia wyboru elementu podstawowego jest taka, jak to opisano w poprzedniej sekcji. Macierze **A**, **W** i **Z** zapisuje się jako macierze blokowe:

$$\mathbf{A}=\begin{bmatrix} a_{11} & \mathbf{a}_{1*}^T & a_{1n} \\ \mathbf{a}_{*1} & \hat{\mathbf{A}} & \mathbf{a}_{*n} \\ a_{n1} & \mathbf{a}_{n*}^T & a_{nn} \end{bmatrix}=\begin{bmatrix} 1 & \mathbf{0}^T & 0 \\ \mathbf{w}_{*1} & \hat{\mathbf{W}} & \mathbf{w}_{*n} \\ 0 & \mathbf{0}^T & 1 \end{bmatrix}\begin{bmatrix} z_{11} & \mathbf{z}_{1*}^T & z_{1n} \\ 0 & \hat{\mathbf{Z}} & 0 \\ z_{n1} & \mathbf{z}_{n*}^T & z_{nn} \end{bmatrix}=$$

$$\begin{bmatrix} z_{11} & \mathbf{z}_{1*}^T & z_{1n} \\ \mathbf{w}_{*1}z_{11}+\mathbf{w}_{*n}z_{n1} & \mathbf{w}_{*1}\mathbf{z}_{1*}^T+\hat{\mathbf{W}}\hat{\mathbf{Z}}+\mathbf{w}_{*n}\mathbf{z}_{n*}^T & \mathbf{w}_{*1}z_{1n}+\mathbf{w}_{*n}z_{nn} \\ z_{n1} & \mathbf{z}_{n*}^T & z_{nn} \end{bmatrix}=\mathbf{WZ},$$

gdzie $\hat{\mathbf{W}}$ i $\hat{\mathbf{Z}}$ są macierzami kwadratowymi o postaci odpowiednio **W** i **Z** (ale o rozmiarze o dwa mniejszym niż **A**), zaś $\hat{\mathbf{A}}$ jest pełną macierzą kwadratową. $\mathbf{a}_{1*}^T$, $\mathbf{a}_{n*}^T$, $\mathbf{z}_{1*}^T$ oraz $\mathbf{z}_{n*}^T$ to wektory poziome (o n–2 elementach), a $\mathbf{a}_{*1}$, $\mathbf{a}_{*n}$, $\mathbf{w}_{*1}$, $\mathbf{w}_{*n}$ – wektory pionowe (także mające n–2 elementów). Przez porównanie odpowiednich elementów uzyskano:

$$\begin{aligned} &a_{11}=z_{11};\quad a_{1n}=z_{1n};\quad a_{n1}=z_{n1};\quad a_{nn}=z_{nn} \\ &\mathbf{a}_{1*}^T=\mathbf{z}_{1*}^T;\quad \mathbf{a}_{n*}^T=\mathbf{z}_{n*}^T; \\ &\begin{cases}\mathbf{a}_{*1}=\mathbf{w}_{*1}z_{11}+\mathbf{w}_{*n}z_{n1}\\ \mathbf{a}_{*n}=\mathbf{w}_{*1}z_{1n}+\mathbf{w}_{*n}z_{nn}\end{cases} \\ &\hat{\mathbf{A}}=\mathbf{w}_{*1}\mathbf{z}_{1*}^T+\hat{\mathbf{W}}\hat{\mathbf{Z}}+\mathbf{w}_{*n}\mathbf{z}_{n*}^T. \end{aligned} \tag{6}$$

Korzystając ze wzorów (6) można nieformalnie zapisać algorytm w następujący sposób:

1. Przypisać pierwszy wiersz macierzy **A** pierwszemu wierszowi macierzy **Z**.

Przypisać ostatni wiersz macierzy **A** ostatniemu wierszowi macierzy **Z**.

Wyznaczyć $\mathbf{w}_{*1}$ oraz $\mathbf{w}_{*n}$ wyliczone z układu równań:

$$\mathbf{w}_{*1}\leftarrow\alpha\mathbf{a}_{*n}-\beta\mathbf{a}_{*1},\quad \alpha=\frac{z_{n1}}{z_{1n}z_{n1}-z_{11}z_{nn}},\quad \beta=\frac{z_{nn}}{z_{1n}z_{n1}-z_{11}z_{nn}}$$

$$\mathbf{w}_{*n}\leftarrow\gamma\mathbf{a}_{*1}-\delta\mathbf{a}_{*n},\quad \gamma=\frac{z_{1n}}{z_{1n}z_{n1}-z_{11}z_{nn}},\quad \delta=\frac{z_{11}}{z_{1n}z_{n1}-z_{11}z_{nn}}.$$

Zaktualizować wewnętrzną podmacierz macierzy **A** o rozmiarze n–2 (bez pierwszego i ostatniego wiersza i bez pierwszej i ostatniej kolumny):

$$\hat{\mathbf{A}} \leftarrow \hat{\mathbf{W}}\hat{\mathbf{Z}} = \hat{\mathbf{A}} - \mathbf{w}_{*1}\mathbf{z}_{1*}^T - \mathbf{w}_{*n}\mathbf{z}_{n*}^T$$

$n \leftarrow n-2$

Jeżeli $n>2$, to wrócić do punktu 1.

Po wykonaniu powyższego algorytmu wyznaczone już są macierze **W** i **Z** postaci (2). Pozostaje tylko rozwiązać równanie postaci **Wc**=**b**. Równanie to można zapisać macierzowo-wektorowo:

$$\begin{bmatrix} 1 & \mathbf{0}^T & 0 \\ \mathbf{w}_{*1} & \hat{\mathbf{W}} & \mathbf{w}_{*n} \\ 0 & \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ \hat{\mathbf{c}} \\ c_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \hat{\mathbf{b}} \\ b_n \end{bmatrix}, \tag{7}$$

gdzie $\hat{\mathbf{W}}$ oznacza macierz kwadratową o rozmiarze o dwa mniejszym od poprzedniego, a $\mathbf{w}_{*1}$, $\mathbf{w}_{*n}$, $\hat{\mathbf{c}}$, $\hat{\mathbf{b}}$ oznaczają wektory pionowe o rozmiarze o dwa mniejszym od wyjściowego. Korzystając z zapisu (7) można napisać algorytm wyznaczania wektora **c**:

1. $c_1 \leftarrow b_1$

$\hat{\mathbf{b}} \leftarrow \hat{\mathbf{b}} - \mathbf{w}_{*1}b_1 - \mathbf{w}_{*n}b_n$

$c_n \leftarrow b_n$

$n \leftarrow n-2$.

Jeżeli $n>2$, to wrócić do punktu 1.

Układ równań **Zx**=**c** rozwiązuje się klasycznie, tzn. od środka rozwiązując układy równań 2×2 i aktualizując wektor **b**.

Przy użyciu notacji *MATLAB*a [4] zapisano poniżej blokowo-punktowy algorytm rozwiązywania równania **Ax**=**b** w następującej postaci dla macierzy o rozmiarze parzystym (nie ma straty ogólności – dla rozmiaru nieparzystego drobna różnica jest tylko przy rozwiązywaniu układu **Zx**=**c**):

```
% pętla eliminacji – kroki redukcji:
for k = 0:m
  k2 = n-k-1
% tu wybór elementu podstawowego...
% ...zamiana wierszy...
% ...i zamina elementow wektora b
  det = A(k2, k)*A(k, k2) - A(k, k)*A(k2, k2);
  alfa=A(k2, k)/det;
  beta=A(k2, k2)/det;
  gama=A(k, k2)/det;
  delta=A(k, k)/det;
```

```
% wyznaczanie współczynników macierzy W
  W(k+1:k2-1;k)=alfa* A(k+1:k2-1,k2)-beta* A(k+1:k2-1,k);
  W(k+1:k2-1;k2)=gama* A(k+1:k2-1,k)-delta* A(k+1:k2-1,k2);
% uaktualnienie macierzy A
  A(k+1:k2-1,k+1:k2-1)=A(k+1:k2-1,k+1:k2-1)- W(k+1:k2-1;k)
*A(k,k+1:k2-1)- W(k+1:k2-1;k2)* A(k2,k+1:k2-1);
% uaktualnienie b
  b(k+1:k2-1) = b(k+1:k2-1) + wk(k+1:k2-1)*b(k)
+wk2(k+1:k2-1)*b(k2)
% znajdowanie rozwiazań x
for j = m:0
% rozwiązywanie układu równań 2x2
  j2 = n - j + 1
  det = A(j, j)*A(j2, j2) - A(j, j2)*A(j2, j)
  x(j) = (b(j)*A(j2, j2) - b(j2)*A(j, j2))/det
  x(j2) = (b(j2)*A(j, j) - b(j)*A(j2, j))/det
% uaktualnienie b(1:j-1)
  b(1:j-1) = b(1:j-1)-x(j)*A(1:j-1,j) - x(j2)* A(1:j-1,j2)
% uaktualnienie b(n-j+2:n)
  b(n-j+2:n)=b(n-j+2:n)-x(j)*A(n-j+2:n,j)-x(j2)
*A(n-j+2:n,j2)
```

Dla potrzeb niniejszego artykułu przedstawiony algorytm oznaczono przez **B2WZ**, jego złożoność czasowa jest rzędu $O(n^3)$.

5. Implementacja i eksperyment numeryczny

Algorytmy **AWZ** i **B2WZ** zostały zaimplementowane w języku *C* [6] dla liczb rzeczywistych o pojedynczej precyzji. Programy skompilowano przy użyciu dostępnego kompilatora języka *C* pod Linuxa (*gcc*), dodatkowo **B2WZ** przy kompilacji miał dołączone potrzebne biblioteki *BLAS1* i *BLAS2* (*Basic Linear Algebra Subprograms*, korzystano z biblioteki *ATLAS* [3], która jest implementacją *BLAS*a automatycznie dostrajającą się do architektury maszyny). *BLAS1* i *BLAS2* to zestaw podprogramów umożliwiających wykonywanie podstawowych obliczeń algebry liniowej. *BLAS1* zawiera operacje typu wektor-wektor, zaś *BLAS2* zawiera operacje macierz-wektor. Przy wyznaczaniu współczynników macierzy **W** użyto funkcji *axpy* ($y \leftarrow y + \alpha x$) oraz *scal* ($y \leftarrow \alpha y$), zaś uaktualnienie macierzy **A**

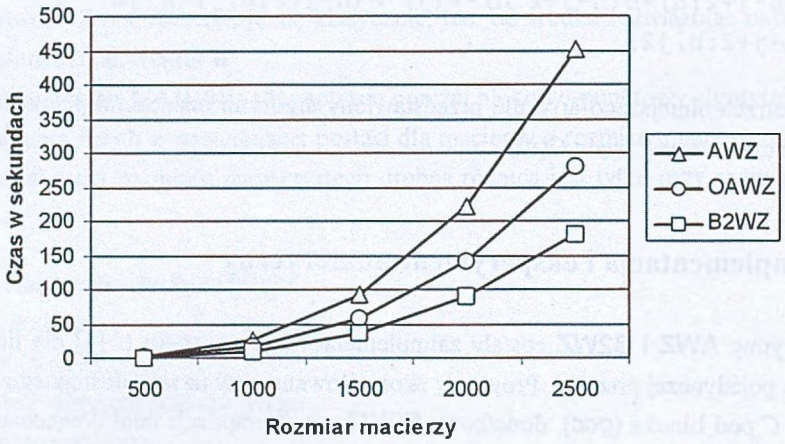
wymagało funkcji biblioteki *BLAS2 syr* ($A \leftarrow A + \alpha x y^T$). Dodatkowo w części drugiej algorytmu – przy znajdowaniu rozwiązania x – użyto funkcji *axpy* ($y \leftarrow y + \alpha x$) do uaktualniania wektora b.

Każdy z algorytmów był kompilowany na dwa sposoby: z wykorzystaniem wbudowanej optymalizacji algorytmu przez kompilator (`gcc -O3`) oraz bez tej optymalizacji (`gcc -O0`). W przypadku algorytmu **B2WZ** optymalizacja przez kompilator nie zmienia wyników, stąd nie bierze się jej w poniższych omówieniach pod uwagę. Natomiast wyniki algorytmu **AWZ** skompilowanego z opcją optymalizacji oznaczane są na wykresach przez **OAWZ**, ponieważ różnią się wyraźnie od nieoptymalizowanych.

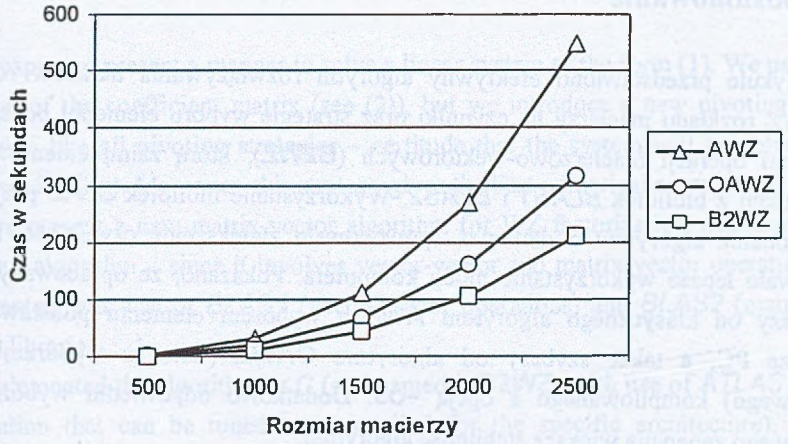
Powyższy algorytm testowano na trzech różnych komputerach PC (512 MB pamięci RAM każdy):

1) dwuprocesorowym Pentium III 733 MHz (rys. 1);
2) jednoprocesorowym Pentium III 667 MHz (rys. 2);
3) jednoprocesorowym Pentium IV 1.4 GHz (rys. 3).

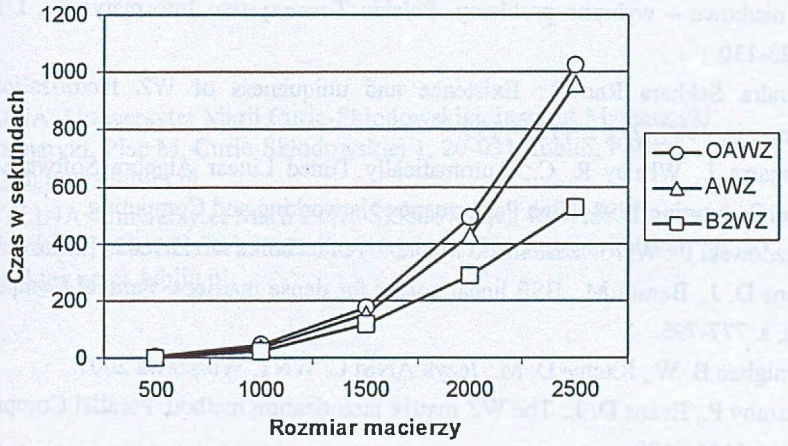
Oba algorytmy uruchamiano dla macierzy nieosobliwych. Otrzymane rezultaty są prezentowane na wykresach na rys. 1–3. Można zauważyć, że przedstawiony algorytm **B2WZ** wykonuje się najkrócej i szybkość jego działania jest najwyższa.



Rys. 1. Czas wykonywania badanych algorytmów dla macierzy różnych rozmiarów na komputerze o dwóch procesorach Pentium III 733 MHz
Fig. 1. Time performance of analyzed algorithms for matrices of various sizes – a dual Pentium III 733 MHz processor machine



Rys. 2. Czas wykonywania badanych algorytmów dla macierzy różnych rozmiarów na komputerze o jednym procesorze Pentium III 667 MHz
Fig. 2. Time performance of analyzed algorithms for matrices of various sizes – a Pentium III 667 MHz processor machine



Rys. 3. Czas wykonywania badanych algorytmów dla macierzy różnych rozmiarów na komputerze o jednym procesorze Pentium 4 1.4 GHz
Fig. 3. Time performance of analyzed algorithms for matrices of various sizes – a Pentium 4 1.4 GHz processor machine

6. Podsumowanie

W artykule przedstawiono efektywny algorytm rozwiązywania układów równań (1) metodą WZ rozkładu macierzy na czynniki oraz strategię wyboru elementu podstawowego przy użyciu operacji macierzowo-wektorowych (**B2WZ**), którą zaimplementowano przy użyciu funkcji z bibliotek *BLAS1* i *BLAS2*. Wykorzystanie bibliotek *BLAS* przyspieszyło czas wykonania algorytmu dodatkowo, spowodowało zwiększenie czytelności kodu oraz spowodowało lepsze wykorzystanie mocy komputera. Pokazano, że opracowany algorytm jest szybszy od klasycznego algorytmu **AWZ** z wyborem elementu podstawowego na komputerze PC, a także szybszy od algorytmu **OAWZ** (także z wyborem elementu podstawowego) kompilowanego z opcją `-O3`. Dodatkowo odpowiedni wybór elementu podstawowego zapewnia większą stabilność algorytmu.

LITERATURA

1. Bylina B., Bylina J., Stpiczyński P.: Blokowo-punktowy rozkład WZ macierzy. Obliczenia naukowe – wybrane problemy. Polskie Towarzystwo Informatyczne, Lublin 2003, s. 123-130.
2. Chandra Sekhara Rao S.: Existence and uniqueness of WZ factorization. Parallel Computig, 1997 (23), s. 1129-1139.
3. Dongarra J., Whaley R. C.: Automatically Tuned Linear Algebra Software (ATLAS). SuperComputing 1998: High Performance Networking and Computing.
4. Drozdowski P.: Wprowadzenie do Matlaba. Politechnika Krakowska, Kraków 1996.
5. Evans D. J., Barulli M.: BSP linear solver for dense matrices. Parallel Computing, 1998 (24), s. 777-795.
6. Kernighan B. W., Ritchie D. M.: Język ANSI C. WNT, Warszawa 2001.
7. Yalamov P., Evans D. J.: The WZ matrix factorization method. Parallel Computing, 1995 (21), s. 1111-1120.

Recenzent: Prof. dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 18 sierpnia 2003 r.

Abstract

In this paper we present a manner to solve a linear system of the form (1). We use the WZ factorization of the coefficient matrix (see (2)), but we introduce a new pivoting strategy, which gives – like all pivoting strategies – certitude that the system will be solved (if the matrix is not singular). Moreover, this strategy gives the best numerical correctness.

Next we present a new matrix-vector algorithm for WZ factorization with our pivoting strategy. Our algorithm – since it involves vector-vector and matrix-vector operations – can be implemented with use of *BLAS1* (vector-vector operations) and *BLAS2* (matrix-vector operations) libraries.

We implemented the algorithm in *C* (and named it **B2WZ**) with use of *ATLAS* (a *BLAS* implementation that can be tuned and compiled for the specific architecture). Next we compare (using different machines) its performance with the performance of the traditional (not matrix-vector) algorithm with pivoting (**AWZ**) and with the traditional one with pivoting compiled with built-in compiler optimization on (**OAWZ**). The results (displayed in figures 1-3) show that our algorithm and implementation is much faster than both **OAWZ** and **AWZ**.

Adresy

Beata BYLINA: Uniwersytet Marii Curie-Skłodowskiej, Instytut Matematyki, Zakład Informatyki, Plac M. Curie-Skłodowskiej 1, 20-031 Lublin, Polska, beatas@golem.umcs.lublin.pl.

Jarosław BYLINA: Uniwersytet Marii Curie-Skłodowskiej, Instytut Matematyki, Zakład Informatyki, Plac M. Curie-Skłodowskiej 1, 20-031 Lublin, Polska, jmbylina@hektor.umcs.lublin.pl.