NAUKI PODSTAWOWE
CZASOPISMO TECHNICZNE
TECHNICAL TRANSACTIONS
FUNDAMENTAL SCIENCES
WYDAWNICTWO POLITECHNIKI KRAKOWSKIEJ
1-NP/2009
ZESZYT 8
ROK 106
ISSUE 8
YEAR 106

SERGIY FIALKO*

BLOKOWA WIELOFRONTALNA METODA PODSTRUKTUR DO ROZWIĄZYWANIA DUŻYCH UKŁADÓW RÓWNAŃ MES

THE BLOCK SUBSTRUCTURE MULTIFRONTAL METHOD FOR SOLUTION OF LARGE FINITE ELEMENT EQUATION SETS

Streszczenie

W niniejszym artykule przedstawiono metodę bezpośrednią rozwiązywania dużych układów równań liniowych algebraicznych z macierzami rzadkimi symetrycznymi, które powstają z zastosowaniem metody elementów skończonych do zadań mechaniki konstrukcji. W przeciwieństwie do znanych implementacji metody wielofrontalnej, podana metoda bazuje na agregacji podstruktur, na które została podzielona konstrukcja – na podstawie algorytmu uporządkowania, z jednoczesną eliminacją całkowicie zagregowanych równań. Tym sposobem faktoryzacja źródłowej macierzy rzadkiej o dużym rozmiarze doprowadzona zostaje do faktoryzacji sekwencji gęstych macierzy o stosunkowo niewielkim wymiarze. W artykule skoncentrowano się przede wszystkim na osiągnięciu wysokiej wydajności na wielordzeniowych komputerach PC.

Słowa kluczowe: *macierz rzadka*, *wielowątkowość*, *wektoryzowanie obliczeń*, *faktoryzacja*

Abstract

The direct method for solution of the large linear algebraic equation sets with sparse symmetrical matrices, which arise during application of the finite element method to the problems of structural mechanics, is presented. Proposed approach differs from well-known realizations of the multifrontal method because it is based on the assembling of substructures on which the whole structure is divided by means of application of reordering algorithm, with simultaneous elimination of the fully assembled equations. Thus the factorization of the source large sparse matrix is reduced to factorization of the series of dense matrices with respectively small dimension. The main attention in this article is paid to achievement of high performance on multi-core desktop computers.

Keywords: *sparse matrix, multithreading, vectorization of computations, factoring*

* Dr hab. inż. Sergiy Fialko, Instytut Modelowania Komputerowego, Wydział Fizyki, Matematyki i Informatyki Stosowanej, Politechnika Krakowska.

1. Wstęp

Obecnie rozmiar modeli obliczeniowych metody elementów skończonych do zadań mechaniki konstrukcji osiąga 2–4 mln równań, przy czym większość projektantów preferuje rozwiązywanie tych zadań na komputerach PC.

Współczesny komputer PC przeważnie ma dwu- lub czterordzeniowy procesor, którego rdzenie (dalej będziemy je nazywać procesorami) dzielą wspólną pamięć główną. Słabym punktem takiej budowy jest niska przepustowość magistrali, która musi jednocześnie obsłużyć kilka procesorów. Z tego zaś wynika, że na komputerach wielordzeniowych przy zrównolegleniu istotnie mogą być przyspieszone tylko pewne typy algorytmów. Zwykle są to algorytmy poziomu 3 BLAS (*Basis Linear Algebra Subroutines*).

Drugą charakterystyczną cechą komputerów PC jest stosunkowo niewielka objętość pamięci głównej. Wymaga to od oprogramowania MES raczej oszczędnego jej użytkowania. W pierwszej kolejności dotyczy to metod przeznaczonych do rozwiązywania układów równań MES. Przy braku wymaganej pojemności pamięci wykonuje się zrzut informacji na dysk, ponieważ w wypadku metody frontalnej [7] i wielofrontalnej [2] macierz frontalna o największym rozmiarze powinna być umieszczona w pamięci głównej. Dlatego też w kontekście zadań z macierzami symetrycznymi bardzo ważne jest zastosowanie symetrycznego sposobu przechowywania danych.

Wysoką wydajność osiaga się najczęściej dzięki użyciu bibliotek BLAS, Intel Math Kernal Library (Intel MKL), IMSL i innych. Jednak w tych bibliotekach obliczenie dopełnienia Schura – nawet dla macierzy symetrycznych – wymagałoby jego przechowywania w postaci ogólnej, co wiązałoby się z umieszczeniem N^2 elementów. Odbywa się to w celu osiągnięcia maksymalnej wydajności. Jednym z zadań niniejszego artykułu jest utworzenie takiego algorytmu obliczania dopełnienia Schura, który – z jednej strony – będzie korzystał z symetrycznego schematu przechowywania i będzie potrzebował $N\cdot(N + 1)/2 \approx N^2/2$ elementów macierzy, a z drugiej – pozwalał na obliczenia na wysokim poziomie wydajności. Dotyczy to faktoryzacji macierzy gęstej.

W celu realizacji powyższych zadań opracowano własne procedury wysokiej wydajności dla symetrycznego schematu przechowywania macierzy.

2. Blokowa wielofrontalna metoda podstruktur

Najpopularniejszą i najczęściej używaną metodą rozwiązywania dużych układów równań liniowych algebraicznych z macierzami rzadkimi w oprogramowaniu MES jest metoda wielofrontalna [8]. Klasyczna metoda wielofrontalna to metoda algebraiczna. Oznacza to, że w tym wypadku informację źródłową stanowi zagregowana macierz układu równań liniowych algebraicznych, umieszczona w formacie skompresowanym.

W niniejszym artykule przedstawiono blokową metodę wielofrontalną podstruktur, składającą się z następujących etapów:

– uporządkowanie numerów węzłów modelu obliczeniowego – informacją źródłową jest graf przyległości węzłów, a celem takiego uporządkowania jest zmniejszenie liczby elementów niezerowych macierzy, powstających na pozycjach elementów zerowych przy faktoryzacji. Takie elementy nazwiemy zapełnieniami;
– cała konstrukcja jest podzielona na oddzielne elementy skończone. Zaczynamy zatem zbierać tę konstrukcję tak, żeby zabezpieczyć kolejność eliminacji węzłów, podaną

przez algorytm uporządkowania. Eliminowanie węzła oznacza, że wszystkie równania skojarzone z tym węzłem zostaną wyeliminowane;

– proces faktoryzacji przedstawiamy jako procedurę „krok po kroku”, w której na każdym kroku będzie dokonywana agregacja kolejnych podstruktur stworzonych w poprzednich krokach tego procesu, tak by wszystkie węzły przeznaczone do wyeliminowania na kroku podanym były całkowicie złożone. Całkowicie złożonymi będziemy nazywać takie węzły, dla których współczynniki równań nie ulegają zmianie przy dodawaniu dowolnej z pozostałych podstruktur. Takie równania można natychmiast wyeliminować, nie czekając na zakończenie procesu agregacji. Aby węzeł był całkowicie złożony, wystarczy połączyć wszystkie elementy skończone i podstruktury, które go zawierają.

Dla płyty kwadratowej z siatką 2 × 2 (rys. 1a)) podano etapy faktoryzacji dla kolejności eliminowania węzłów 1, 3, 7, 9, 2, 6, 8, 4, 5. W pierwszej kolejności rozdzielamy całą konstrukcję na oddzielne elementy skończone (rys. 1b)). Węzły 1, 3, 7, 9 eliminujemy odpowiednio na poziomie każdego z elementów skończonych 1, 2, 3, 4, ponieważ są to węzły całkowicie złożone. Dla elementu skończonego nr 1 macierz sztywności ma postać

$$\mathbf{K}_1 = \begin{array}{c} \begin{matrix} 5 & 4 & 2 & 1 \end{matrix} \\ \begin{pmatrix} \mathbf{A}_{11} & & & \\ \mathbf{A}_{21} & \mathbf{A}_{22} & & \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} & \\ \mathbf{A}_{41} & \mathbf{A}_{42} & \mathbf{A}_{43} & \mathbf{A}_{44} \end{pmatrix} \end{array} \begin{matrix} 5 \\ 4 \\ 2 \\ 1 \end{matrix} \tag{1}$$

Numery węzłów wchodzących w skład tego elementu skończonego ustawione są z góry. Każdy węzeł zawiera kilka równań, przy czym ich liczba zależy od typu modelu obliczeniowego. Przykładowo, dla modelu przestrzennego, który składa się z prętowych i powłokowych elementów skończonych, w każdym węźle jest 6 równań (3 równania dla stopni swobody translacyjnych i 3 – dla rotacyjnych), jeśli na ten węzeł nie zostały nałożone więzy. Elementami macierzy są więc bloki o rozmiarze 6 × 6. Wewnątrz macierzy używamy numeracji ciągłej – lokalnej. Dla związku z numerami równań macierzy rzadkiej (globalnej) musimy zachować tablicę indeksów tworzoną na podstawie numerów węzłów oznaczonych jak wyżej (zob. (1)).

Węzły umieszczamy w kolejności odwrotnej do podanej przez algorytm uporządkowania: 5, 4, 2, 1. Wtedy bloki dla równań węzła całkowicie złożonego tworzą pasmo w dolnej części macierzy. Ze względu na symetrię przechowujemy w pamięci tylko dolny trójkąt macierzy $\mathbf{K}_1$.

Wykonujemy częściową faktoryzację macierzy blokowej $\mathbf{K}_1$

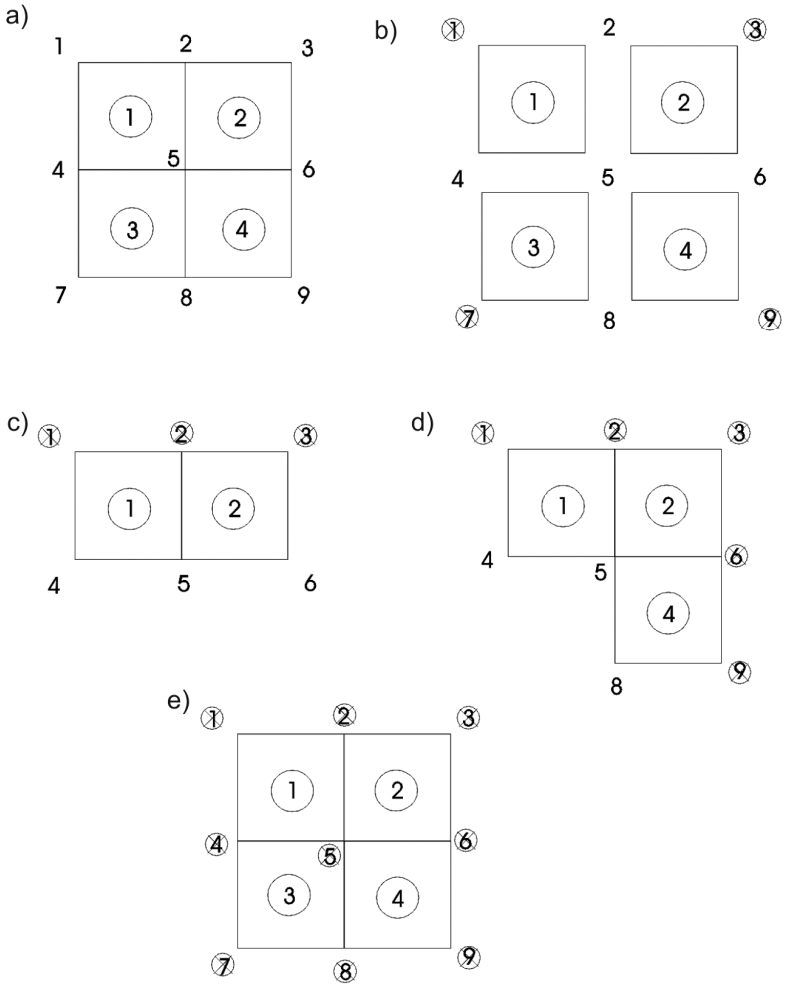
$$\begin{pmatrix} \mathbf{C} & \mathbf{W} \\ \mathbf{W}^T & \mathbf{D} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{C}} & \tilde{\mathbf{W}} \\ 0 & \mathbf{L} \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{I}_S \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ \tilde{\mathbf{W}}^T & \mathbf{L}^T \end{pmatrix}, \tag{2}$$

gdzie

$$\mathbf{C} = \begin{pmatrix} \mathbf{A}_{11} & & \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{pmatrix}, \quad \mathbf{W}^T = \begin{pmatrix} \mathbf{A}_{41} & \mathbf{A}_{42} & \mathbf{A}_{43} \end{pmatrix}, \quad \mathbf{D} = \mathbf{A}_{44}.$$

Przedstawiamy faktoryzację blokową w następujący sposób

$$\begin{aligned} &1.\quad \mathbf{D} = \mathbf{L}\cdot\mathbf{I}_S\cdot\mathbf{L}^T \quad \rightarrow \quad \mathbf{L}, \mathbf{I}_S \\ &2.\quad \mathbf{W}^T = \mathbf{L}\cdot\mathbf{I}_S\cdot\tilde{\mathbf{W}}^T \rightarrow \quad \tilde{\mathbf{W}} \\ &3.\quad \mathbf{C} = \tilde{\mathbf{C}} + \tilde{\mathbf{W}}\cdot\mathbf{I}_S\cdot\tilde{\mathbf{W}}^T \rightarrow \tilde{\mathbf{C}} = \mathbf{C} - \tilde{\mathbf{W}}\cdot\mathbf{I}_S\cdot\tilde{\mathbf{W}}^T \end{aligned} \tag{3}$$



Rys. 1. Etapy łączenia podstruktur przy faktoryzacji macierzy blokową wielofrontalną metodą podstruktur
Fig. 1. Stages of substructure assembling for matrix factoring by means of block multifrontal substructure method

Wzory (3) wynikają z (2) przy mnożeniu odpowiednich macierzy klatkowych. Najpierw sfaktoryzujemy dolny diagonalny blok **D** i otrzymamy dolną trójkątną macierz **L** i przekątne znaków $\mathbf{I}_S$ (punkt 1 w (3)). Przekątna znaków jest macierzą diagonalną, której

przekątna składa się z 1 lub –1. Pozwala to uogólnić faktoryzacje Choleskiego na macierze ujemnie określone.

Dalej znajdziemy klatkę $\mathbf{W}^T$ (punkt 2 z (3)), musimy zatem rozwiązać układ równań liniowych algebraicznych z macierzą dolną trójkątną **L** i wykonać diagonalne skalowanie. Natomiast najbardziej pracochłonną procedurą jest obliczanie dopełnienia Schura (punkt 3 z (3)).

Po zakończeniu faktoryzacji częściowej klatki $\tilde{\mathbf{W}}, \mathbf{L}$ są częściami macierzy sfaktoryzowanej globalnej. Te dane umieszczamy w buforze pamięci B1, który przy wypełnieniu zostanie zapisany na dysku. Klatka **C** (niezakończony front) będzie potrzebna przy kolejnych krokach faktoryzacji – przechowujemy ją w drugim buforze B2, przy czym dla dużych zadań, kiedy będzie brakować miejsca w pamięci głównej, ten bufor również zostanie zapisany na dysk.

Umieszczenie całkowicie zagregowanych równań w dolnej części macierzy wyklucza wolną procedurę przesunięcia bloków macierzy tworzących niezakończony front. Jest to istotne przy podnoszeniu wydajności algorytmu faktoryzacji. Usuwanie z macierzy gęstych (dalej będziemy je nazywać macierzami frontalnymi) całkowicie zagregowanych równań ogranicza ich wzrost [3–5].

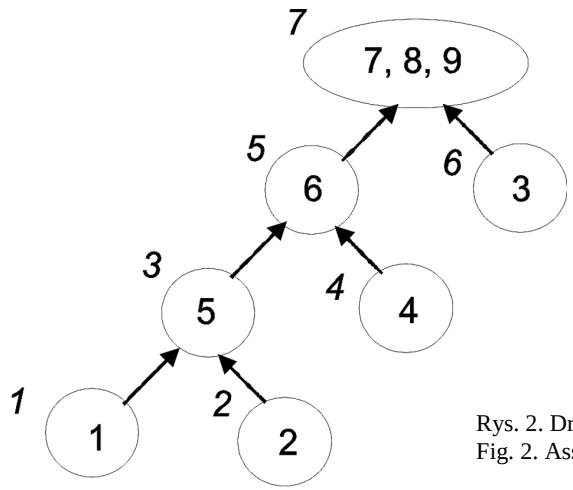
Po faktoryzacji macierzy elementów skończonych 1–4 węzły 1, 3, 7, 9 zostaną wyeliminowane. Otaczamy te węzły kołem, a następnie przekreślamy (rys. 1b)). W dalszej kolejności trzeba wyeliminować węzeł 2. W tym celu łączymy podstruktury z poprzednich kroków eliminacji, które zawierały ten węzeł. Są to elementy skończone 1, 2 po wyeliminowaniu węzłów – odpowiednio – 1, 3 (rys. 1c)). Łączenie podstruktur odpowiada agregowaniu nowej macierzy frontalnej z niezakończonych frontów tych podstruktur. Znów otrzymamy gęstą macierz, w której część równań będzie całkowicie zagregowana, a pozostała część będzie tworzyć niezakończony front. Do macierzy tej stosujemy przedstawioną powyżej procedurę częściowej blokowej faktoryzacji i przesyłamy klatki $\tilde{\mathbf{W}}, \mathbf{L}$ do buforu B1, a w buforze B2 powstaje wtedy niezakończony front, który składa się z węzłów 5, 4, 6.

W celu eliminacji węzła 6 łączymy poprzednią podstrukturę z podstrukturą elementu 4 (rys. 1d)), otrzymujemy zatem kolejną gęstą macierz i stosujemy blokową faktoryzację. Następnie łączymy ostatnią podstrukturę z podstrukturą elementu 3 (rys. 1e)). Tu wszystkie pozostałe węzły są całkowicie złożone, stosujemy zatem blokową faktoryzację tej macierzy.

W trakcie niektórych kroków agregacji okazuje się, że po łączeniu podstruktur z poprzednich kroków powstaje kilka całkowicie złożonych węzłów. W naszym przykładzie taką sytuację można zobaczyć na rys. 1e): węzły 5, 4, 8 są gotowe do eliminacji. Na takich krokach zwiększamy rozmiar bloku całkowicie zagregowanych równań i zamiast 3 sekwencyjnych kroków eliminacji równań węzłów 8, dalej 4, i w końcu 5 o rozmiarze bloków 6, wykonujemy jeden krok eliminacji o rozmiarze bloku $3 \cdot 6 = 18$.

Podział macierzy gęstej na bloki jest kluczowym momentem zwiększania wydajności procedury faktoryzacji [1, 5, 6]. Im większy jest rozmiar bloku M całkowicie zagregowanych równań, tym większą wydajność można osiągnąć. Eksperymenty numeryczne pokazują, że wydajność rośnie do granicy $M \approx 80$. Dlatego na krokach, które dopuszczają zwiększenie rozmiaru bloku wskutek łączenia grupy całkowicie złożonych węzłów, zawsze zwiększamy te bloki pod warunkiem, że $M \leq 80$.

Jeśli każdą z podstruktur przedstawi się wierzchołkami grafu, a relacje pomiędzy nimi – krawędziami, to powstaje drzewo agregacji podstruktur (rys. 2). Jako relacje rozumiemy tutaj sekwencje agregacji podstruktur pomiędzy krokami.



Rys. 2. Drzewo agregacji-eliminacji podstruktur
Fig. 2. Assembling-elimination tree

W dalszej kolejności przyjmiemy, że każda podstruktura bieżącego kroku razem z odpowiednią macierzą gęstą, listą indeksów globalnych i listą podstruktur *s* poprzednich kroków, które tworzą podstrukturę kroku bieżącego, reprezentuje obiekt klasy C++ typu *front*. Tym samym drzewo agregacji-eliminacji podstruktur będziemy nazywać drzewem frontalnym. Drzewo frontalne przypomina drzewo eliminacji klasycznej metody wielofrontalnej [2], jednak traktuje się je zupełnie inaczej.

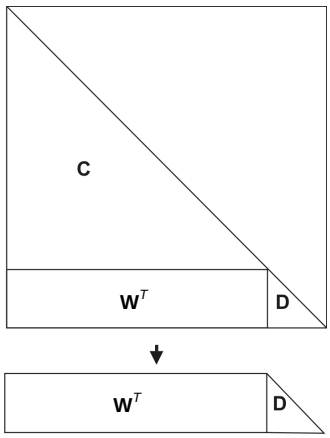
W celu zmniejszenia objętości pamięci głównej, potrzebnej do przechowywania niezakończonych frontów, wprowadzamy ponowne numerowanie frontów. Nowe numery na rys. 2 zapisano kursywą. Ostatni front powstaje przy łączeniu frontów sekwencyjnych 7, 8, 9, służących do eliminowania węzłów – odpowiednio – 8, 4, 5. Szczegółowe informacje na ten temat można znaleźć w pracach [3–5]. Następnie skoncentrujemy się na opracowaniu metody obliczenia dopełnienia Schura (punkt 3 (3)) pod kątem zrównoleglenia na podstawie wielowątkowości, wektoryzowania obliczeń, blokowania na poziomie rejestrów wektorowych i pakowania danych w celu zmniejszenia liczby błędów przy odczycie danych z pamięci podręcznej każdego procesora.

3. Faktoryzacja macierzy frontalnej

Podstawy matematyczne częściowej faktoryzacji macierzy frontalnej przedstawiono za pomocą wzorów (1)–(3).

Przy symetrycznym sposobie przechowywania dolny trójkąt macierzy frontalnej jest umieszczony w pamięci głównej kolumna po kolumnie, przy czym całkowicie zagregowana część macierzy – bloki $\mathbf{W}^T$, $\mathbf{D}$ – znajduje się w dolnej części. Przemieszczamy te bloki do buforu B1 (rys. 3) i wykonujemy kroki 1, 2 faktoryzacji blokowej (3).

Faktoryzacja bloków $\mathbf{W}^T$, $\mathbf{D}$ odbywa się w adresach pamięci bufora B1. Następnie pakujemy macierz $\tilde{\mathbf{W}}$ tak, żeby zminimalizować liczbę błędów przy odczycie pamięci podręcznej ze względu na użycie 128-bitowych XMM rejestrów. Współczesne komputery PC na platformie ia32 mają dostęp do 8 XMM rejestrów, a na platformie Intel 64 – do 16. Każdy XMM rejestr jest w stanie, w ramach jednego cyklu procesora, wykonać 2 mnożenia lub 2 dodawania dla liczb typu *double*, z których każda zajmuje 64 bity. Na tym polega wektoryzowanie obliczeń.



Rys. 3. Struktura macierzy frontalnej
Fig. 3. Structure of frontal matrix

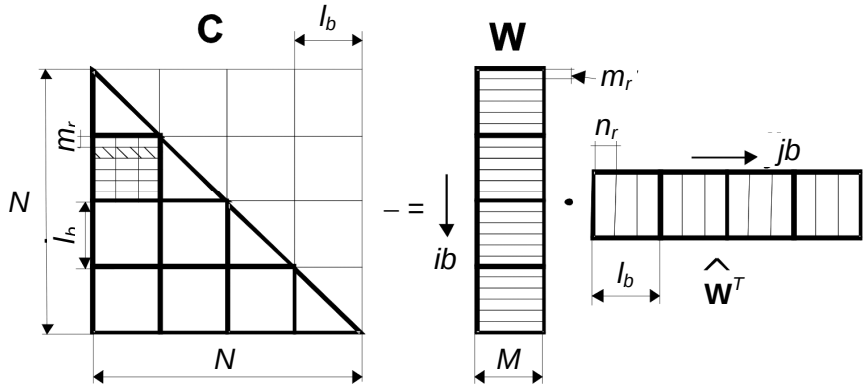
W celu zwiększenia wydajności algorytmu na poziomie „pamięć podręczna–rejestry procesora" zastosujemy schemat blokowania XMM rejestrów [6] $n_r \times m_r$ (rys. 5). Dla platformy ia32 $n_r = 2$, $m_r = 4$, a dla platformy Intel 64 – $n_r = m_r = 4$.

Algorytm obliczenia dopełnienia Schura przedstawiono na rys. 4. Zrównoleglenie wykonujemy na podstawie OpenMP. Aby zwiększyć skalowalność (zdolność do przyspieszenia przy zwiększeniu liczby procesorów), zrównoleglenie wprowadzimy również do etapu 2 wzoru (3). Jedyną częścią sekwencyjną przy faktoryzacji macierzy frontalnej pozostaje obliczenie bloku diagonalnego $\mathbf{L}$ (etap 1 wzoru (3)).

```
// Pack W̃
# pragma omp parallel for private(ib, jb) scheduler(dynamic)
for( jb = 0; jb < Nb; jb++)
{
   // Pack   Ŵ^T_jb
   for(ib = jb;ib < Nb;ib++)
   {
      C̃_{ib,jb} = C̃_{ib,jb} − W̃_ib Ŵ^T_jb
   }
}
```

Rys. 4. Algorytm obliczenia dopełnienia Schura
Fig. 4. Schur complement evaluation algorithm

W tym wypadku $\hat{\mathbf{W}}_{jb}^{T} = \mathbf{I}_S \cdot \left(\tilde{\mathbf{W}}_{jb}\right)^{T}$. Podział na bloki dla macierzy $\tilde{\mathbf{W}}$, $\hat{\mathbf{W}}^{T}$, $\mathbf{C}$ podano na rys. 5. Rozmiar M istotnie wpływa na wydajność algorytmu i zależy od tego, ile całkowicie złożonych węzłów udało się połączyć na danym kroku faktoryzacji.

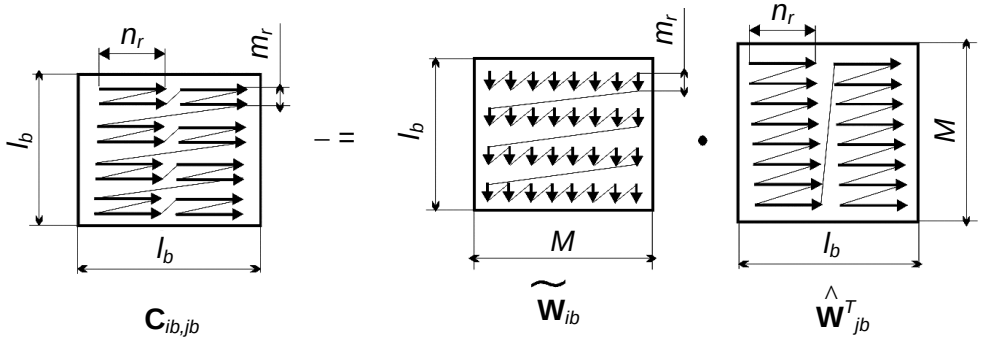


Rys. 5. Podział macierzy $\tilde{\mathbf{W}}$, $\tilde{\mathbf{W}}^{T}$, $\mathbf{C}$ na bloki. Linie grube – blokowanie pamięci podręcznej, linie cienkie – blokowanie rejestrów $m_r \times n_r$

Fig. 5. Subdivision of matrices $\tilde{\mathbf{W}}$, $\tilde{\mathbf{W}}^{T}$, $\mathbf{C}$ by blocks. The heavy line – cache blocking, thin lines – register's blocking $m_r \times n_r$

Rozmiar l_b jest wyznaczono z warunku, według którego w pamięci podręcznej $L1$ powinny się znajdować trzy bloki: blok $m_r \times M$ macierzy $\tilde{\mathbf{W}}_{ib}$, blok $l_b \times M$ macierzy $\hat{\mathbf{W}}_{jb}^{T}$ i blok $m_r \times l_b$ macierzy $\mathbf{C}_{ib,\,jb}$. Z tego wynika: $l_b = \dfrac{L1 - m_r M}{M + m_r}$, gdzie $L1$ to objętość pamięci podręcznej $L1$.

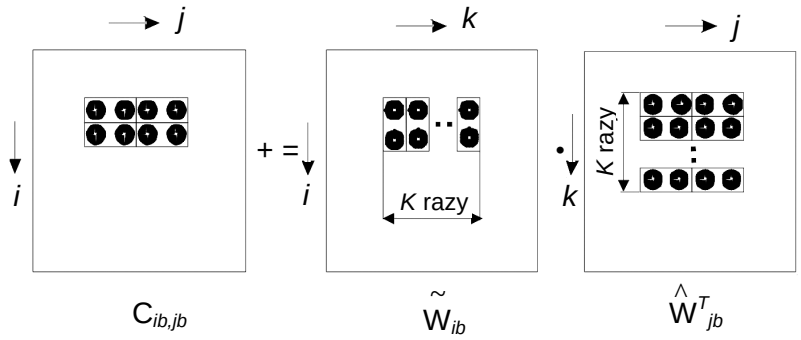
Pakowanie danych dla odpowiednich bloków [6] przedstawiono na rys. 6.



Rys. 6. Pakowanie danych dla bloków macierzy $\tilde{\mathbf{W}}$, $\hat{\mathbf{W}}^{T}$, $\mathbf{C}$

Fig. 6. Pack of data for blocks of matrices $\tilde{\mathbf{W}}$, $\hat{\mathbf{W}}^{T}$, $\mathbf{C}$

Schemat blokowania XMM rejestrów dla platformy 32-bitowej podano na rys. 7. Rozładowanie rejestrów wektorowych w blok $\mathbf{C}_{ib,jb}$ wykonujemy tak, jak podano na rys. 6. Po zakończeniu obliczeń bloku $\mathbf{C}_{ib,jb}$ przepakowujemy dane, aby macierz **C** była umieszczona kolumna po kolumnie.



Rys. 7. Schemat blokowania XMM rejestrów $n_r \times m_r$ z rozwijaniem pętli wewnętrznej K razy
Fig. 7. Register's blocking diagram $n_r \times m_r$. The unroll of internal loop is produced K times

4. Wyniki

Wszystkich obliczeń w wymienionych przykładach dokonano na komputerze Intel® Core™2 Quad CPU Q6600 @2.40 GHz, pamięć główna DDR2 800 MHz 8 GB. System operacyjny Windows Vista Business 64 umożliwia uruchomienie 32-bitowych aplikacji (platforma ia32), a także 64-bitowych (platforma Intel 64). Przy uruchamianiu 32-bitowych aplikacji największy rozmiar buforu, który można zaalokować dla aplikacji GDI (*Graphic Device Interface*), wynosi ok. 1,4 GB, przy czym dostępnych jest 8 rejestrów XMM ($n_r = 2$, $m_r = 4$ lub $n_r = 4$, $m_r = 2$). Przy uruchamianiu 64-bitowych aplikacji największy rozmiar buforu jest ograniczony tylko rozmiarem fizycznym pamięci głównej, liczba dostępnych XMM rejestrów wynosi 16, dlatego istnieje możliwość zwiększenia rozmiaru bloku ($n_r = m_r = 4$). Powoduje to wzrost wydajności w stosunku do 32-bitowych aplikacji.

4.1. Test dla macierzy gęstej

Wydajność i skalowalność faktoryzacji macierzy frontalnej sprawdzimy na macierzy gęstej o rozmiarze $N = 4\,800$ przy rozmiarze bloku całkowicie zagregowanych równań $M = 48$. Dla tego zadania wykonano 4800/48 = 100 kroków faktoryzacji, przy czym każdy krok przedstawiono wzorami (2) i (3). Wyniki pokazano w tabeli 1. Przyspieszenie obliczeń przy zwiększeniu liczby procesorów opisano za pomocą współczynnika S_p. Jest to stosunek czasu obliczeń T_1 na jednym procesorze do czasu obliczeń T_p na p procesorach.

Aby wyjaśnić wpływ wektoryzowania obliczeń przedstawionych w niniejszym artykule, w tabeli 2 pokazano wyniki, gdzie zastosowano tylko blokowanie pamięci podręcznej.

Chociaż blokowanie pamięci podręcznej (podział macierzy $\tilde{\mathbf{W}}$, $\hat{\mathbf{W}}^T$, $\mathbf{C}$ na zgrubne bloki, oznaczone na rys. 5 grubymi liniami) wystarczy, aby otrzymać dobrą skalowalność

(na 4 procesorach otrzymaliśmy ponad 3-krotne przyspieszenie), wektoryzowanie obliczeń zgodnie z podaną techniką powoduje zwiększenie wydajności o 2,75–3,3 razy.

Tabela 1

Wydajność faktoryzacji macierzy frontalnej (blokowanie XMM rejestrów, blokowanie pamięci podręcznej)

Platforma	ia32				Intel 64 (×64)			
Liczba procesorów	1	2	3	4	1	2	3	4
MFLOPS	4 459	8 395	11 567	14 299	5 526	10 437	14 559	18 151
$S_p=T_1/T_p$	1	1,88	2,59	3,21	1	1,89	2,63	3,28

Tabela 2

Wydajność faktoryzacji macierzy frontalnej (blokowanie tylko pamięci podręcznej)

Platforma	ia32				Intel 64 (x64)			
Liczba procesorów	1	2	3	4	1	2	3	4
MFLOPS	1 575	3 100	4 519	5 231	1 587	3 120	4 599	5 513
$S_p=T_1/T_p$	1	1,97	2,87	3,32	1	1,97	2,90	3,47

4.2. Płyta kwadratowa z siatką 400 × 400

Rozważmy przypadek płyty kwadratowej stalowej $E = 2 \cdot 10^{11}$ Pa, $\nu = 0{,}3$ o rozmiarze 1 m szerokości i 0,01 m grubości, gdzie: E – moduł Younga, ν – współczynnik Poissona. Dwa wierzchołki płyty są sztywno zamocowane, a do trzeciego przykładamy siłę skupioną $Nx = Ny = N\,z = 1000$ N. Do modelowania tego zadania wykorzystujemy 4-węzłowy powłokowy element skończony, który ma 6 stopni swobody w węźle. Czas faktoryzacji macierzy solverem BSMFM (*block substructure multifrontal method*) porównujemy z czasem faktoryzacji macierzy solverem *sparse direct* programu ANSYS 11.0 (tab. 3). Rozmiar zadania wynosi 964 794 równań.

Solver BSMFM nie ustępuje w szybkości solverowi ANSYS 11.0.

Tabela 3

Czas faktoryzacji macierzy globalnej dla płyty kwadratowej 400 × 400 (964 794 równań) [s]

Liczba procesorów	Platforma ia32		Platforma Intel 64
	ANSYS v. 11.0	BSMFM	BSMFM
1	221	117	86
2	176	90	60
4	159	78	51

4.3. Płytka kwadratowa z siatką 800 × 800

Dla podanego w poprzednim przykładzie zadania zagęścimy siatkę. Liczba równań wynosi wtedy 3 849 594. W tabeli 4 przedstawiono wyniki porównania.

Solver ANSYS 11.0 na 32-bitowej platformie nie rozwiązał tego problemu z powodu braku pamięci głównej, ale problem ten udało się rozwiązać solverem BSMFM.

Następne przykłady reprezentują modele obliczeniowe przygotowane w programie SCAD (www.scadsoft.com). Jest to jeden z najczęściej używanych programów MES

do obliczeń konstrukcji budowlanych na terenie Wspólnoty Niezależnych Państw, dostosowany do norm projektowania tego regionu.

Tabela 4

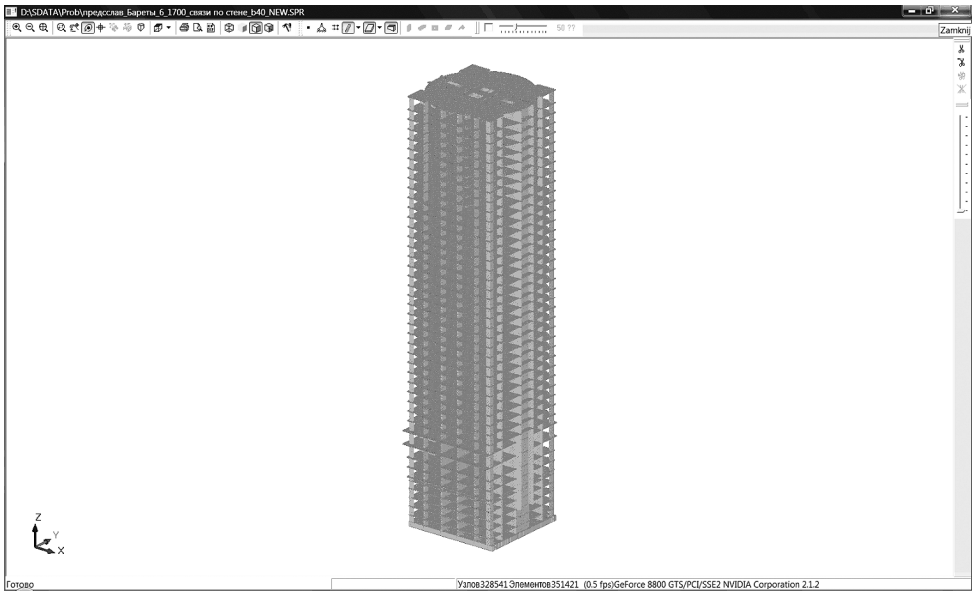
Czas faktoryzacji macierzy globalnej dla płyty kwadratowej 800 × 800 (3 849 594 równań) [s]

Liczba procesorów	Platforma ia32		Platforma Intel 64
	ANSYS v. 11.0	BSMFM	BSMFM
1	Brak pamięci	978	888
2	Brak pamięci	768	551
4	Brak pamięci	670	460

4.4. Model budynku wielopiętrowego (1 956 634 równań)

Model przedstawiono na rys. 8. Rozmiar sfaktoryzowanej macierzy wynosi 7,2 GB. Próba rozwiązania tego zadania solverem PARDISO [9] z Intel MKL na platformie Intel 64 zakończyła się niepowodzeniem z powodu niewystarczającej objętości pamięci głównej – wielkość 8 GB okazała się za mała. Za pomocą solvera BSMFM przy 4 procesorach problem ten rozwiązano po 443 s na platformie ia32 i po 413 s na platformie Intel 64.

Ostatnie dwa przykłady dowodzą oszczędności solvera BSMFM, jeśli chodzi o użycie pamięci głównej.



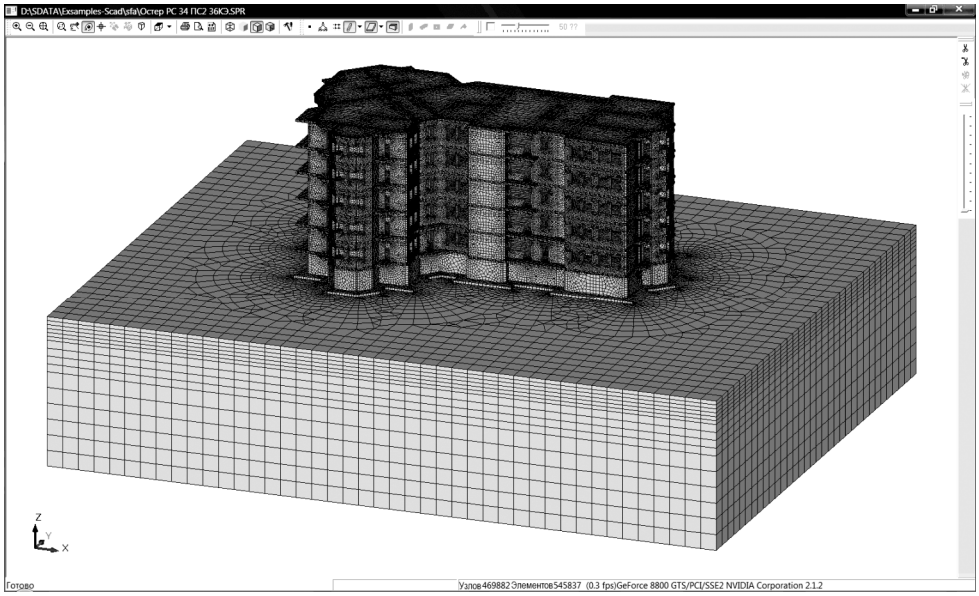
Rys. 8. Model MES budynku wielopiętrowego, 328 541 węzłów, 351 421 elementów skończonych i 1 956 634 równań

Fig. 8. FE model of the multistory building, 328 541 nodes, 351 421 finite elements and 1 956 634 equations

4.5. Zadanie interakcji budynku i gruntu

Ten problem (rys. 9) jest bardzo trudny do wykonania jako zadanie dla solverów bezpośrednich, ponieważ bryła gruntu, modelowana elementami objętościowymi, istotnie zwiększa gęstość macierzy rzadkiej. Dla modelu tworzonego z elementów objętościowych każdy węzeł ma większą liczbę sąsiadów, co powoduje większą liczbę elementów niezerowych w odpowiednich wierszach macierzy globalnej.

Rozmiar zadania wynosi 2 763 181 równań, rozmiar sfaktoryzowanej macierzy – 13 978 MB, a rozmiar największego frontu (największej macierzy gęstej) – 16 542 równania. Ta macierz frontalna zajmuje 1 044 MB pamięci głównej przy symetrycznym schemacie przechowywania (tylko dolny trójkąt macierzy). Dla 32-bitowej wersji solvera BSMFM zabrakło pamięci i tylko 64-bitowa wersja umożliwiła rozwiązanie tego problemu – czas faktoryzacji na 4 procesorach wynosi 19 m 17 s.



Rys. 9. Model MES budynek – grunt zawiera 469 882 węzłów, 545 837 elementów skończonych i 2 763 181 równań

Fig. 9. FE model for soil – structure interaction comprises 469 882 nodes, 545 837 finite elements and 2 763 181 equations

4.6. Różne techniki przyspieszenia solvera BSMFM

Na przykładzie sześcianu z siatką 50 × 50 × 50 (objętościowe elementy skończone o kształcie sześcianu) rozważymy skrócenie czasu obliczeń z zastosowaniem różnych technik przyspieszenia. Jest to typowe zadanie dla testów wydajności solverów bezpośrednich. Niezależnie od tego, że zadanie jest stosunkowo niewielkie (397 941 równań) ten test jest dość wymagający dla solverów bezpośrednich, ponieważ rzadka macierz globalna ma dużą gęstość. Wyniki podano w tab. 5.

W wersji z 2002 roku nie było żadnego blokowania. Faktoryzacja macierzy frontalnej odbywała sie metodą Gaussa (**L · U**) wiersz po wierszu [3].

W wersji z 2004 roku dla faktoryzacji macierzy frontalnej opracowano blokową metodę Choleskiego (**L · S · L**), w której przy obliczeniu dopełnienia Schura na etapie 3 wzoru (3) macierze $\tilde{\mathbf{W}}$, $\hat{\mathbf{W}}^T$, **C** były podzielone na bloki (blokowanie pamięci podręcznej). Jednak rozmiar bloku zależał tylko od liczby równań w węźle modelu obliczeniowego – brakowało łączenia węzłów w celu zwiększenia jego rozmiarów. Na poziome „pamięć podręczna – rejestry procesora" używano algorytmu naiwnego (brak blokowania rejestrów).

Tabela 5

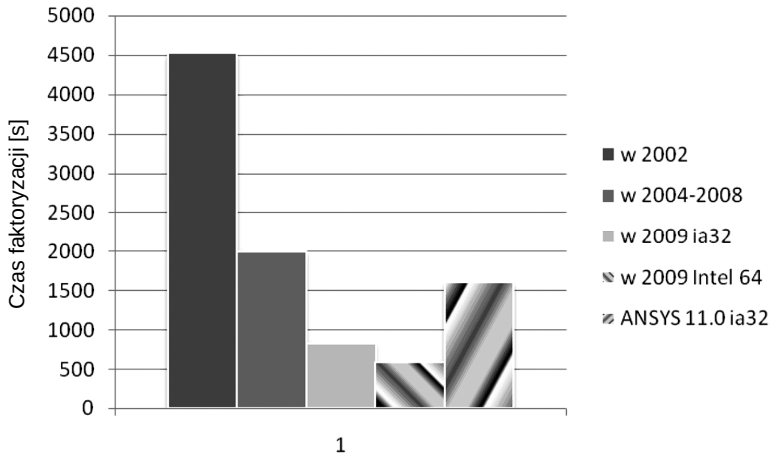
Porównywanie czasu obliczeń różnych wersji solvera BSMFM dla zadania 50 × 50 × 50 [s]

Wersja, rok	Platforma	Liczba procesorów		
		1	2	4
2002	ia32	4 520	–	–
2004	ia32	2 000	–	–
2008	ia32	2 000	1 142	684
2009	ia32	827	504	365
2009	Intel 64	584	322	213

W wersji z 2008 roku użyto zrównoleglenia na podstawie OpenMP (rys. 4, ale bez pakowania bloków macierzy).

W wersji z 2009 roku wprowadzono wektoryzowanie obliczeń z blokowaniem rejestrów XMM i pakowanie danych zgodnie z przyjętym schematem blokowania.

Na rysunku 10 przedstawiono porównanie czasów obliczeń na jednym procesorze.



Rys. 10. Porównywanie czasów obliczeń różnych wersji solvera BSFMF na jednym procesorze. Ostatnia kolumna prezentuje czas rozwiązania zadania solverem sparse direct ANSYS 11.0 na platformie ia32

Fig. 10. Comparison of the computation time for different wersions of BSMFM solver on single processor. The last column presents the sparse direct solver of ANSYS 11.0 for platform ia32

5. Wnioski

Przedstawiona blokowa wielofrontalna metoda podstruktur służąca do rozwiązywania dużych układów równań liniowych algebraicznych, które powstają z stosowaniem metody elementów skończonych do zadań mechaniki konstrukcji, została opracowana dla wielordzeniowych komputerów PC. W odróżnieniu od wiadomych realizacji solvera wielofrontalnego [2], podana metoda jest metodą podstruktur z automatycznym podziałem całej konstrukcji na podstruktury na podstawie wybranego algorytmu uporządkowania. Faktoryzacja macierzy rzadkiej globalnej polega na agregacji podstruktur krok za krokiem z jednoczesnym wyeliminowaniem całkowicie zagregowanych węzłów. Prowadzi to do tego, że faktoryzacja globalnej macierzy rzadkiej redukuje się do częściowej faktoryzacji szeregu macierzy gęstych – frontalnych.

Zastosowanie symetrycznego schematu przechowywania macierzy gęstych w pamięci głównej razem z opracowaniem procedur wysokiej wydajności z wykorzystaniem blokowania pamięci podręcznej, blokowania rejestrów wektorowych, pakowania danych i wielowątkowości umożliwiło osiągnięcie wysokiej wydajności metody w połączeniu z jednoczesnym oszczędnym użyciem pamięci głównej, co jest bardzo ważne dla komputerów PC.

Literatura

[1] Demmel J.W., *Applied Numerical Linear Algebra*, SIAM, Philadelphia 1997.

[2] Amestoy P.R., Duff I.S., L'Excellent J.-Y., *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Comput. Meth. Appl. Mech. Eng. 184, 2000, 501-520.

[3] Fialko S.Yu., *Stress-Strain Analysis of Thin-Walled Shells with Massive Ribs*, Int. App. Mech. 40, N4, 2004, 432-439.

[4] Fialko S.Yu., *A block sparse direct multifrontal solver in SCAD software*, Proceedings of the CMM-2005 – Computer Methods in Mechanics, Częstochowa 2005, 73-74.

[5] Fialko S.Yu., A *Sparse Shared-Memory Multifrontal Solver in SCAD Software*, Proceedings of the International Multiconference on Computer Science and Information Technology, October 20–22, Wisła 2007, 3, 2008, 277-283, (http://www.proceedings2008.imcsit.org/pliks/47.pdf).

[6] Goto K., Van De Geijn R.A., *Anatomy of High-Performance Matrix Multiplication*, ACM Transactions on Mathematical Software, Vol. 34, No. 3, 2008, 1-25.

[7] Scott J.A., *A frontal solver for the 21st century*, Communications in Numerical Methods in Engineering 22, 2006, 1015-1029.

[8] Gould N.I.M., Hu Y., Scott J.A., *A numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations*, Technical report RAL-TR-2005-005, Rutherford Appleton Laboratory, 2005.

[9] Schenk O., Gartner K., *Two-level dynamic scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems*, Parallel Computing 28, 2002, 187-197.