Rozwiązanie układów równań liniowych algebraicznych dla macierzy rzadkich: metody bezpośrednie

- **Metody bezpośrednie – to są takie metody, które za skończoną ilość operacji doprowadzają do wyniku dokładnego, przynajmniej w arytmetyce precyzyjnej.**

- **Dla macierzy rzadkich skuteczność rozwiązania istotnie zależy od:**
  - **zdolności metody uporządkowania zmniejszyć ilość niezerowych elementów w macierzy sfaktoryzowanej**
  - **zdolności solwera umieścić w pamięci głównej duże tablice danych, zabezpieczyć wirtualizacje (podział danych na bloki, z których tylko część znajduje się w pamięci głównej jednocześnie, a inne bloki są pobierane z dysku)**

- **zdolności solwera opracowywać bloki danych, znajdujących się w pamięci głównej, na wysokim poziomie wydajności**

- **Istnieje wielu różnych typów solwerów na dzień dzisiejszy dla macierzy rzadkich symetrycznych**
  - **Solwer skyline – to jest metoda Gaussa lub Choleckiego, w której macierz przechowywana w formacie profilowym. Najbardziej używane metody uporządkowania: RCM, metoda Sloana. To są solwery starego typu, dla dużych macierzy (N > 10 000) wydajność drastycznie spada w skutek powstania znacznej ilości zapełnień. Solwer jest prosty w realizacji, lekko poddaje się wirtualizacji.**

  - **Solwer frontalny dla MES. Macierz źródłowa nigdy nie agregowana w postaci jawnej. Nośnikami informacji o elementach macierzy są macierzy elementów skończonych – macierzy nie wielkiego rozmiaru. Proces faktoryzacji: po koleje pozostaje wprowadzone asemblowanie macierzy elementów skończonych, wykrycie kompletnie zebranych równań (to są takie równania, współczynniki których nie zmieniają się przy doładowaniu macierzy dowolnych z pozostałych elementów skończonych – takie równania można wyeliminować, nie doczekując końca procesu asemblowania) i eliminacji tych równań. W taki sposób idzie równoległe asemblowanie – eliminacje równań, przy czym eliminacje przez cały czas**

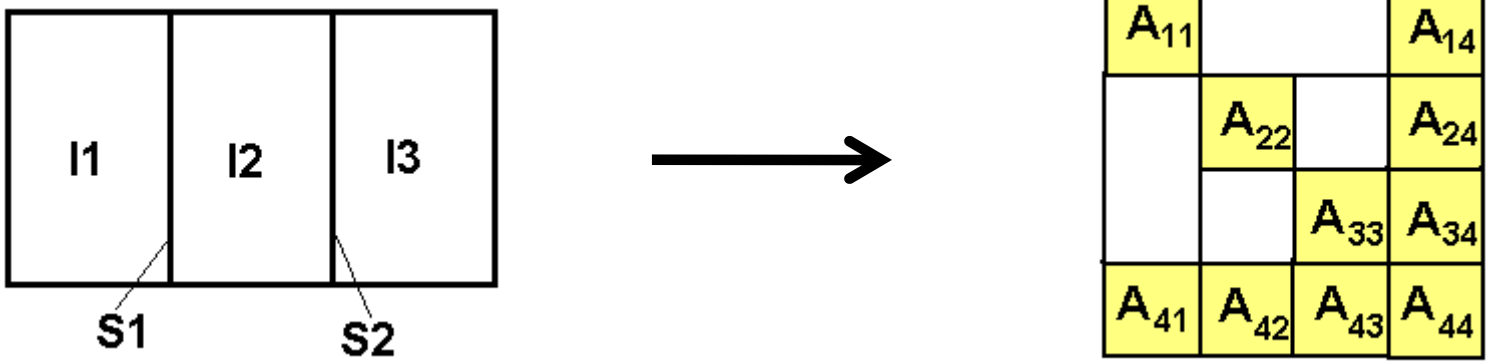
jest wykonywana w macierzy gęstej stosownie nie wielkiego rozmiaru – tak zwanej macierzy frontalnej. Sfaktoryzowane części macierzy od razu pozostają wypchane na dysk, dalej będzie asemblowana macierz kolejnego elementu skończonego, i tak dopóki faktoryzacja się nie skończy. Sfaktoryzowana macierz powstaje w całości na dysku. Zamiast uporządkowania węzłów lub równań jest wykonywane uporządkowanie grafu przyległości elementów skończonych w celu minimalizacji rozmiaru macierzy frontalnej. Najbardziej używane metody uporządkowania – RCM, Sloan. Przy takim podejściu frontalna macierz będzie tylko jedną.

o **Solver domain decomposition – macierz ma specyficzną blokowo-diagonalną strukturą z obramowaniem w skutek rozdzielenia obszaru obliczeniowego na nieprzekrywające podobszary. Taka struktura macierzy jest bardzo łatwą do zrównoleglenia, solwer jest odnośnie prosty w realizacji, lekko poddaje się wirtualizacji, przecież najczęściej takie metody uporządkowania nie są najlepsze w sensie zmniejszenia zapełnieni.**

- **Sparse direct solwer – uporządkowanie najczęściej polega na algorytmie minimalnego stopnia, metodzie włożonych przekrojów, metodach wielopoziomowych. Macierz jest przechowywana w jednym z formatów skompresowanych (na przykład, format kompresowany A. H. Shermana). Nie nadaje się do wirtualizacji. Jest bardzo skuteczne podejście, jeśli macierz może być umieszczona w pamięci głównej.**

- **Solwery multifrontalne – to jest połączenie idei solwera frontalnego z uporządkowaniem dowolnego typu. Jako wynik, jednocześnie powstaje wielu macierzy frontalnych. Nadaje się do wirtualizacji i do zrównoleglenia. Ilość elementów niezerowych w macierzy sfaktoryzowanej jest taka sama, jak i w solwerach sparse. Metoda multifrontalna nadaje możliwość zredukować operacje nad macierzą rzadką do operacji eliminacji nad kolejnością macierzy gęstych stosownie nie wielkiego rozmiary – frontalnych macierzy. Macierz sfaktoryzowana w całości powstaje na dysku. Przynajmniej w MES na dzień dzisiejszy jest najbardziej potężną i wydajną metodą dla rozwiązywania bardzo dużych układów równań (500 000 – 3 000 000 na komputerach klasy PC).**

➢ **Rozważymy solwer domain decomposition z punktu widzenia obliczeń równoległych. Będziemy uważali, że dla wyznaczenia podobszarów była zastosowana metoda przekrojów równoległych: obszar pozostał rozdzielony na podobszary wprowadzeniem separatorów S_1, S_2. Jeśli równania, odniesione do części wewnętrznej podobszarów I_1, I_2, I_3, ustawić przed równaniami, odniesionymi do separatorów – (I_1, I_2, I_3, S_1, S_2), to macierz będzie miała postać blokowo-diagonalną z obramieniem. Bloki diagonalne A_{11}, A_{22}, A_{33} odpowiadają równaniom wewnętrznym I_1, I_2, I_3, a blok A_{44} – równaniom separatorów S_1, S_2. Bloki A41, A42, A43 obramienia powstają dla współczynników równań dla separatorów przy niewiadomych, odniesionych do równań wewnętrznych.**

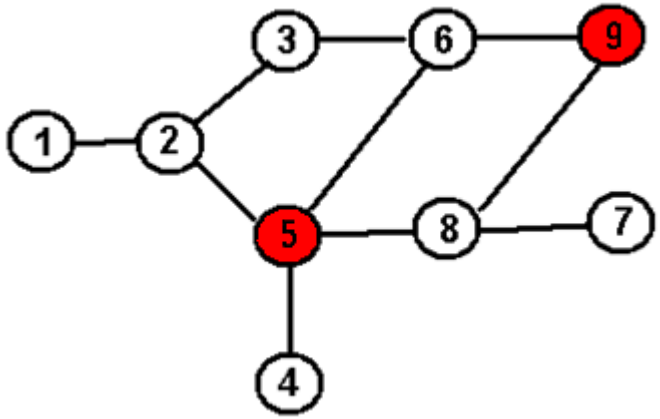


- **Zapełnienia nie pojawią się w blokach, oznaczonych białym kolorem.**

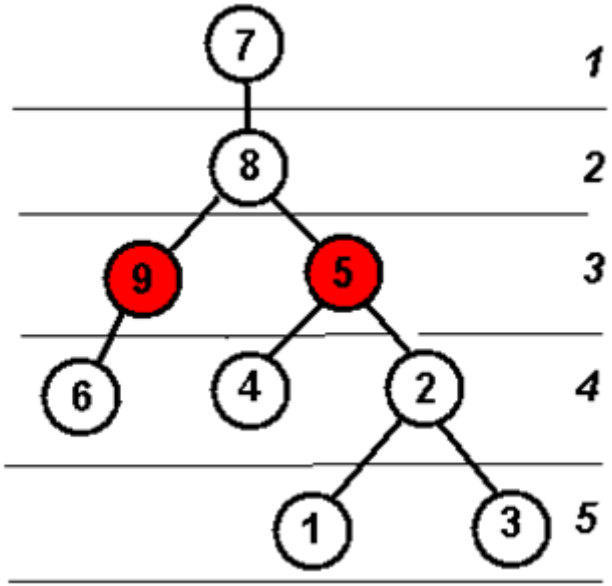
- **Każdy blok diagonalny, oprócz ostatniego, może być sfaktoryzowany niezależnie od innego**

- **Równania wewnętrzne dla każdego bloku diagonalnego uporządkowujemy algorytmem minimalnego stopnia**

- **Przykład:**

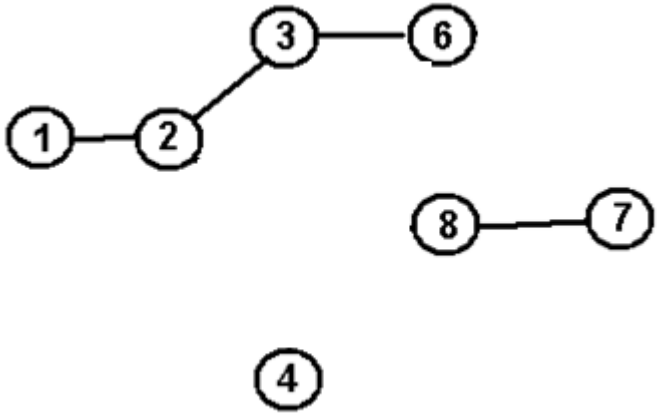


Graf



Struktura poziomów

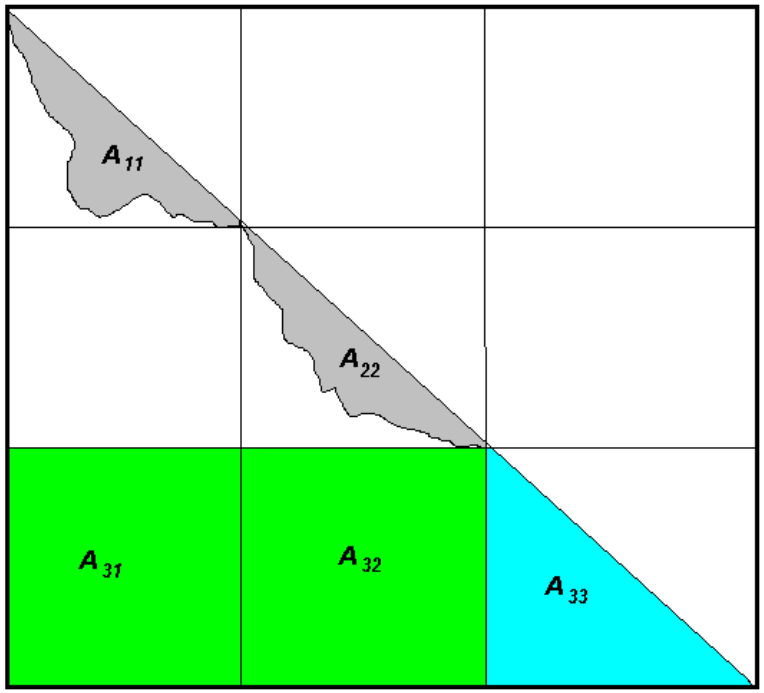
Separator S1 = (9, 5); po usunięciu separatora graf się rozpada na niezależne podgrafy



- **Tablica permutacji będzie taką: Perm(1, 6, 2, 3, 8, 7, 4, 9, 5)**

1		x						
	6		x				x	x
x		2	x					x
	x	x	3				#	#
				8	x		x	x
				x	7		#	#
						4		x
	x		#	x	#		9	#
	x	x	#	x	#	x	#	5

- **Rozwiązanie blokowe:**



Schemat symetryczny

$$for \quad i = 1, 2, ..., \quad n - 1$$

$$A_{ii} = L_{ii} \cdot L_{ii}^{T}$$

$$L_{ii} \cdot L_{ni}^{T} = A_{ni}^{T} \Rightarrow L_{ni}^{T}$$

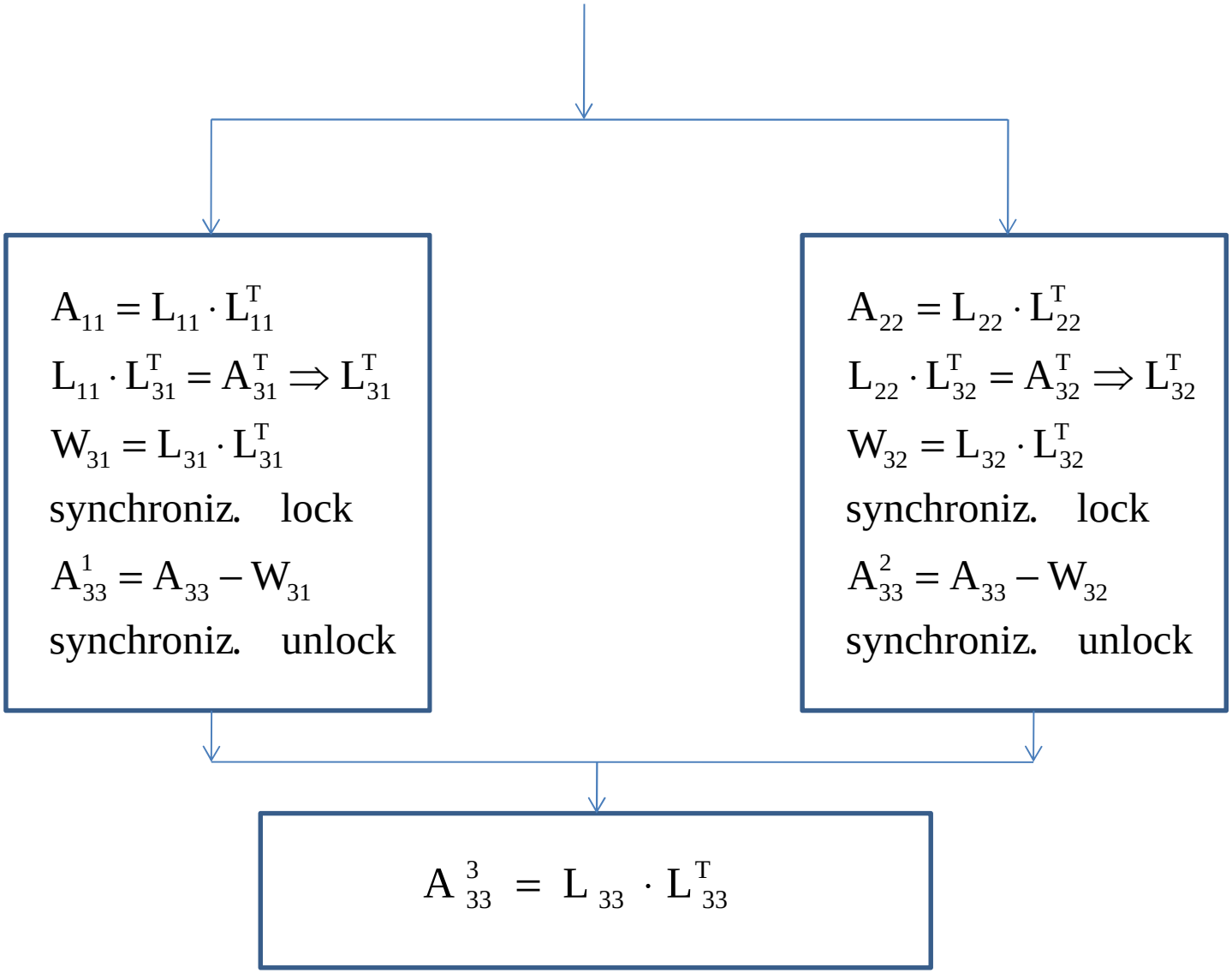
$$W_{ni} = L_{ni} \cdot L_{ni}^{T}$$

$$A_{nn}^{i} = A_{nn} - W_{ni}$$

$$end \quad i \quad loop$$

$$A_{nn}^{n} = L_{nn} \cdot L_{nn}^{T}$$

- **W pętle po i aż do ostatniej instrukcji obliczenia można wykonywać niezależnie od indeksu iteracji i. To nadaje możliwość zrównoleglić operacje. Tylko przy modyfikacji ostatniego diagonalnego bloku jest konieczne zastosowanie synchronizacji.**


$A_{11} = L_{11} \cdot L_{11}^T$
$L_{11} \cdot L_{31}^T = A_{31}^T \Rightarrow L_{31}^T$
$W_{31} = L_{31} \cdot L_{31}^T$
synchroniz. lock
$A_{33}^1 = A_{33} - W_{31}$
synchroniz. unlock
$A_{22} = L_{22} \cdot L_{22}^T$
$L_{22} \cdot L_{32}^T = A_{32}^T \Rightarrow L_{32}^T$
$W_{32} = L_{32} \cdot L_{32}^T$
synchroniz. lock
$A_{33}^2 = A_{33} - W_{32}$
synchroniz. unlock
$A_{33}^3 = L_{33} \cdot L_{33}^T$

- **Schemat asymetryczny: poprawkę do bloku diagonalnego liczymy tak:**

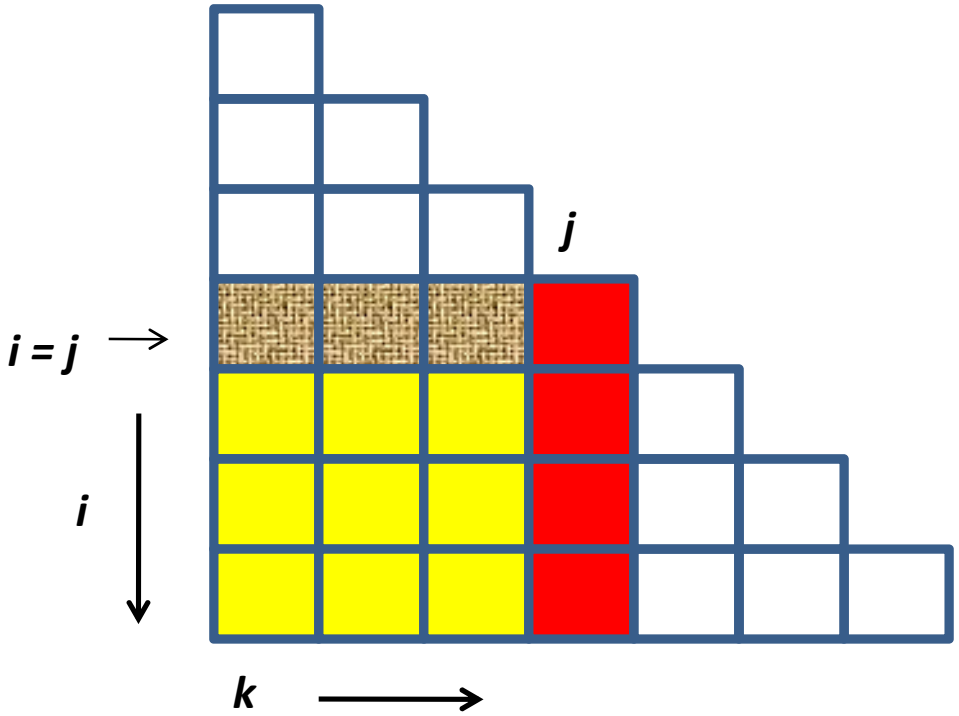
$$W_{ni} = L_{ni} \cdot L_{ni}^T = L_{ni} \cdot L_{ii}^{-1} \cdot A_{ni}^T = A_{ni} \cdot \left[\underbrace{\left(L_{ii}^T\right)^{-1} \cdot \underbrace{L_{ii}^{-1} \cdot A_{ni}^T}_{Z}}_{Y} \right] = A_{ni} \cdot Y$$

$$\begin{aligned}
&for \quad i = 1,2,\ldots,\ n-1 \\
&\quad A_{ii} = L_{ii} \cdot L_{ii}^T \\
&\quad L_{ii} \cdot Z = A_{ni}^T \quad \Rightarrow \quad Z \\
&\quad L_{ii} \cdot Y = Z \quad \Rightarrow \quad Y \\
&\quad W_{ni} = A_{ni} \cdot Y \\
&\quad A_{nn}^{\,i} = A_{nn} - W_{ni} \\
&end \quad i \\
&A_{nn}^{\,n} = L_{nn} \cdot L_{nn}^T
\end{aligned}$$

Zaletą schematu asymetrycznego jest to, że jeden z mnożników macierzowych A_{ni} jest pobrany do faktoryzacji – jest bardziej rzadki od mnożnika L_{ni} schematu symetrycznego (który powstał po faktoryzacji – więc oprócz elementów źródłowych zawiera zapełnienia. Jeśli algorytm mnożenia $A_{ni} \cdot Y$ zorganizować jako dla macierzy rzadkich (obejść elementy zerowe), to ogólna ilość operacji będzie mniej od schematu symetrycznego.

Solwer left-looking

Dla macierzy gęstej:

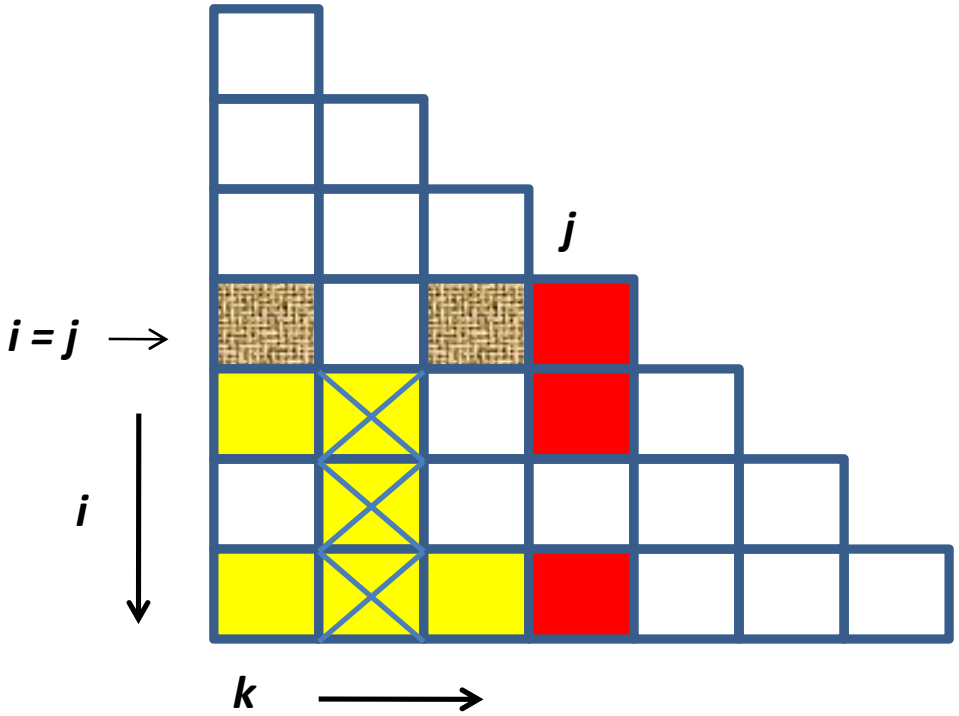


do j=1, N
- ▪ update column j
- do k=1, j-1
  - do i=j, N
    - $a_{ij} = a_{ij} - a_{jk} \cdot a_{ik}$
  - end do
- end do
- ▪ factorize column j
- $l_{jj} = \sqrt{a_{jj}}$
- do i=j+1,N
  - $l_{ij} = a_{ij}/l_{jj}$
- end do

end do

Solwer left-looking sparse

Dla macierzy rzadkiej:



do j=1, N
- update column j

do k ϵ List[j]
do i ϵ L_k
$a_{ij} = a_{ij} - l_{jk} \cdot l_{ki}$
end do
end do
- factorize column j

$l_{jj} = \sqrt{a_{jj}}$
do i ϵ L_j
$l_{ij} = a_{ij}/l_{jj}$
end do
end do

Solwer left-looking sparse

- **Rozważemy ten algorytm na przykładzie:**

$$\begin{pmatrix} 10 & & & & \\ 0 & 10 & & & \\ 1 & 0 & 10 & & \\ 2 & 1 & 0 & 10 & \\ 0 & 1 & 2 & 1 & 10 \end{pmatrix} \rightarrow$$

j=1 (kolumna 1):

$l_{11}=\sqrt{a_{11}} = 3.162;$
$l_{i1}=a_{i1}/l_{11}$

$$\rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 10 & & & \\ 0.316 & 0 & 10 & & \\ 0.632 & 1 & 0 & 10 & \\ 0 & 1 & 2 & 1 & 10 \end{pmatrix}$$

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 10 & & & \\ 0.316 & 0 & 10 & & \\ 0.632 & 1 & 0 & 10 & \\ 0 & 1 & 2 & 1 & 10 \end{pmatrix} \rightarrow$$

j=2 (kolumna 2):

$l_{22}=\sqrt{a_{22}} = 3.162;$
$l_{i2}=a_{i2}/l_{22}$

$$\rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 10 & & \\ 0.632 & 0.316 & 0 & 10 & \\ 0 & 0.316 & 2 & 1 & 10 \end{pmatrix}$$

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 10 & & \\ 0.632 & 0.316 & 0 & 10 & \\ 0 & 0.316 & 2 & 1 & 10 \end{pmatrix}$$

→

j=3 (kolumna 3):

Będzie poprawiona kolumną 1:

$l_{33}= \sqrt{(a_{33}-l_{31}^2)}$; $l_{i3}=(a_{i3} – l_{31}*l_{i1})/l_{33}$;

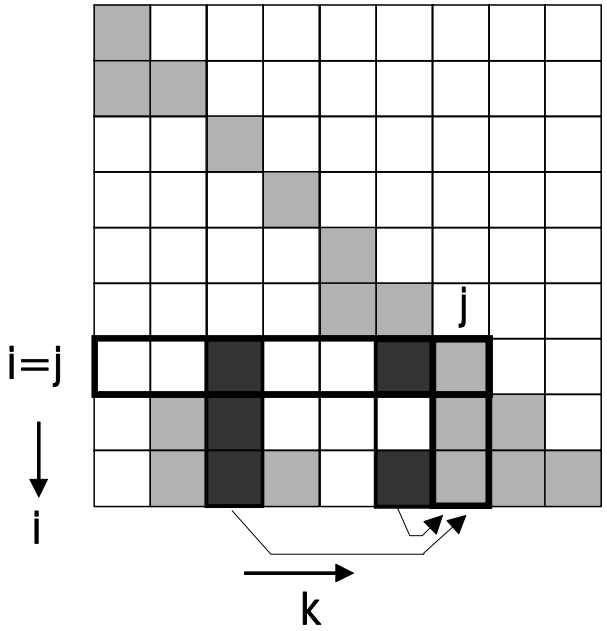
Obliczamy kolumnę j: kolumny, umieszczone od lewej strony, poprawiają kolumnę j, jeśli mają niezerowy element o indeksie l_{jk} :

$$\mathbf{s}_j = \mathbf{s}_j - \mathbf{t}_j, \quad gdzie \quad \mathbf{t}_j = \sum_{k \in List_j} l_{jk} \cdot \mathbf{s}_k,$$

List$_j$ – **zbiór indeksów k, które odpowiadają niezerowym elementom w wiersze i=j**

W wektorze s_k – kolumna *k*, są utrzymane niezerowe elementy i > j (poniżej diagonali)

Poprawienie obejmuje tylko elementy kolumny j, które odpowiadają niezerowym elementom kolumny *k*.



$$\begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 10 & & \\ 0.632 & 0.316 & 0 & 10 & \\ 0 & 0.316 & 2 & 1 & 10 \end{pmatrix} \rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 3.146 & & \\ 0.632 & 0.316 & -0.064 & 10 & \\ 0 & 0.316 & 0.636 & 1 & 10 \end{pmatrix}$$

Kolumna j=4: $l_{44} = \sqrt{(a_{44} - l_{41}^2 - l_{42}^2 - l_{43}^2)} = 3.082$

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 3.146 & & \\ 0.632 & 0.316 & -0.064 & 3.082 & \\ 0 & 0.316 & 0.636 & 1 & 10 \end{pmatrix} \rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 3.146 & & \\ 0.632 & 0.316 & -0.064 & 3.082 & \\ 0 & 0.316 & 0.636 & 0.305 & 10 \end{pmatrix}$$

Kolumna j=5: $l_{55}= \sqrt{(a_{55}-l_{51}^2-l_{52}^2-l_{53}^2 -l_{54}^2)}=3.066$

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 3.146 & & \\ 0.632 & 0.316 & -0.064 & 3.082 & \\ 0 & 0.316 & 0.636 & 0.305 & 3.066 \end{pmatrix}$$

Solwer left-looking sparse – algorytm ogólny

$t \leftarrow 0$ // *wyzerować wektor pomocniczy*

for$(j = 1; j <= N; j++)$ // *obliczamy kolumnę* j

{

$d = 0;$

for$(k \in L_j)$ // L_j *– zbiór numerów kolumn,*

// *dla których* $a_{jk} \neq 0$

{

for$(i \in F_k)$ // *petle po indeksach kolumny* κ

// F_k *– zbiór wartosci indeksu* i*,*

// *dla których* $a_{ik} \neq 0$

{

$t_i = t_i + a_{ik} \cdot a_{jk};$ // a_{jk} *– nie zależa od* i

}

$d = d + a_{jk}^2$

}

$a_{jj} = \sqrt{a_{jj} - d};$ // *obliczamy element na diagonali*

for$(i \in F_k)$

{

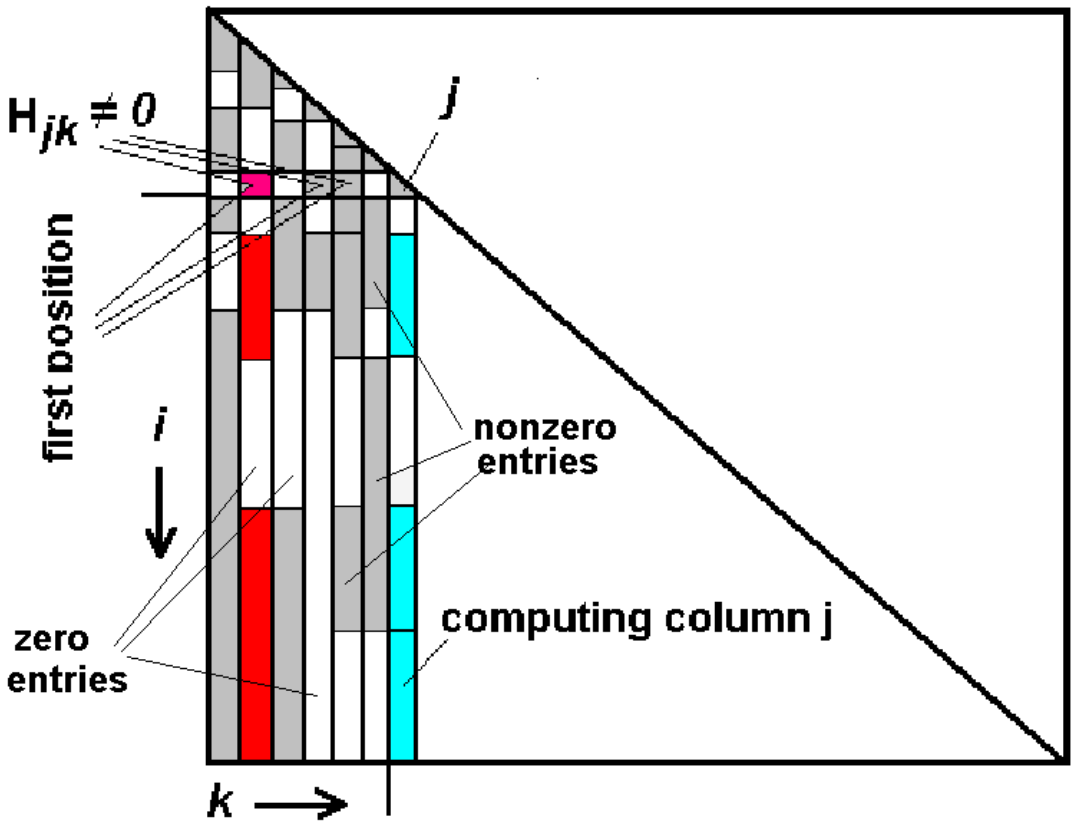
$a_{ij} = (a_{ij} - t_i) / a_{jj};$

$t_i = 0;$ // *wyzerować dla obliczeń kolejnej kolumny*

}

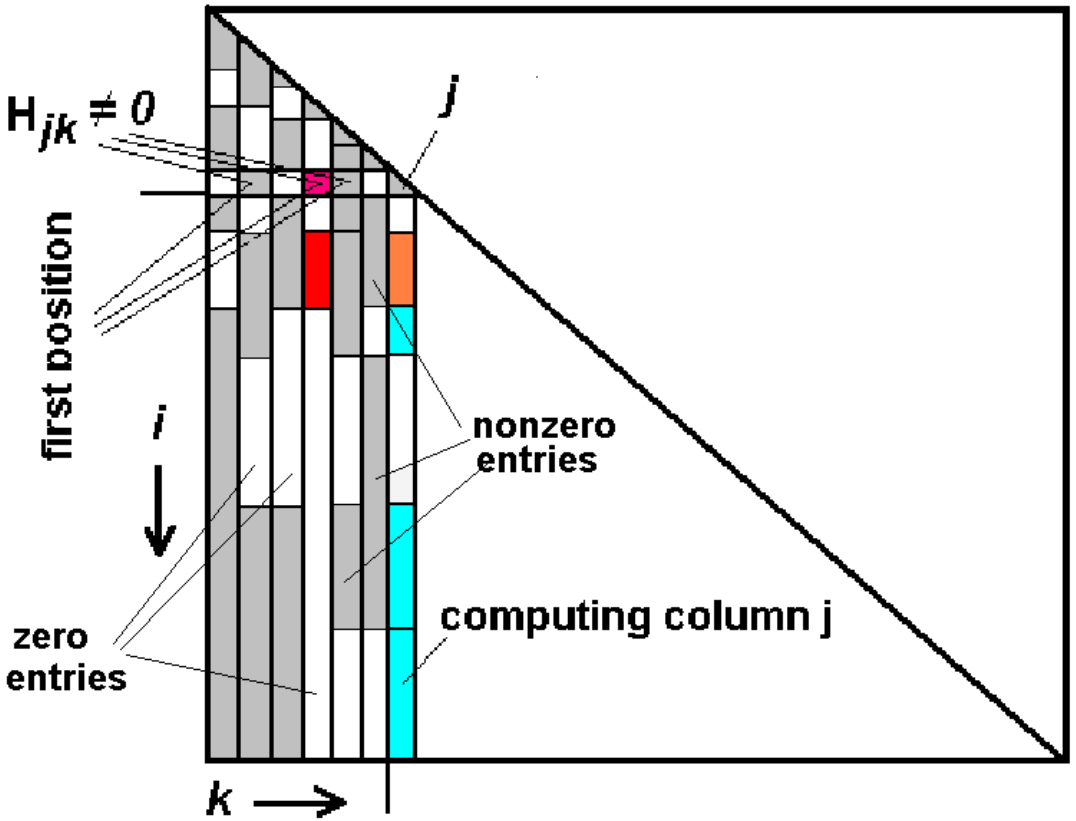
}

1. Sparse technique



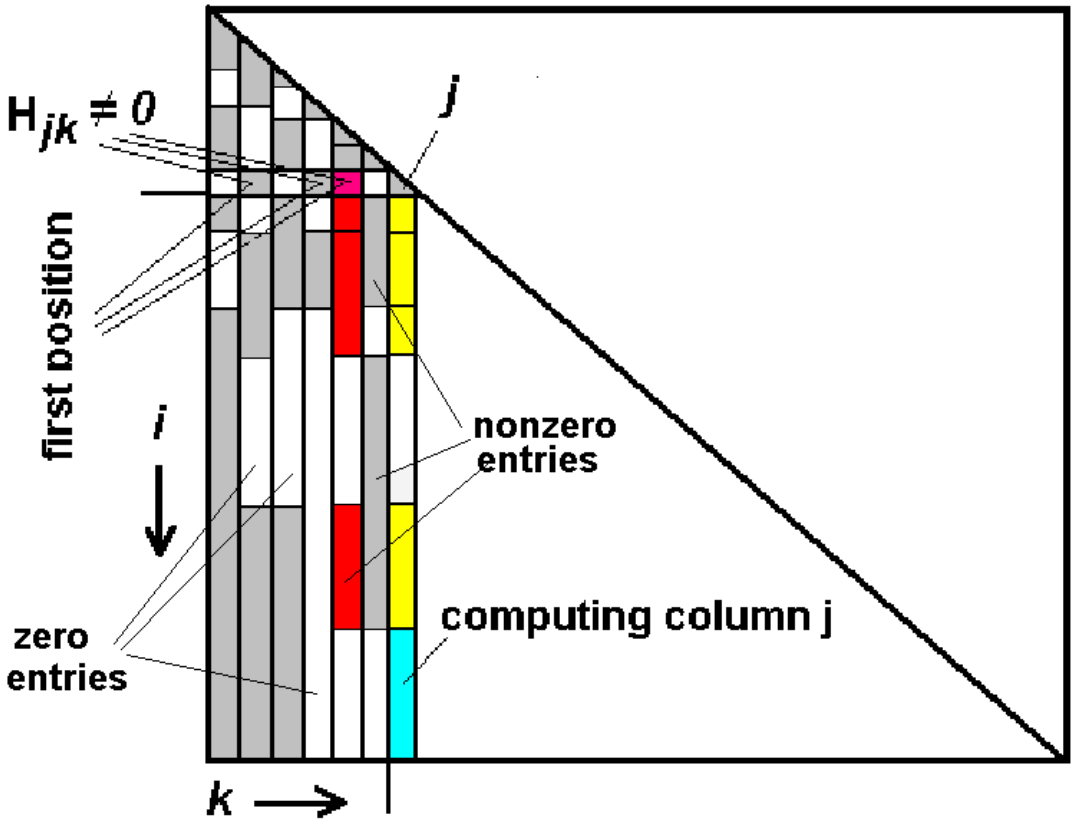
$$\mathbf{t}^j \leftarrow H_{j,2} \cdot \hat{\mathbf{H}}^2$$

1. Sparse technique



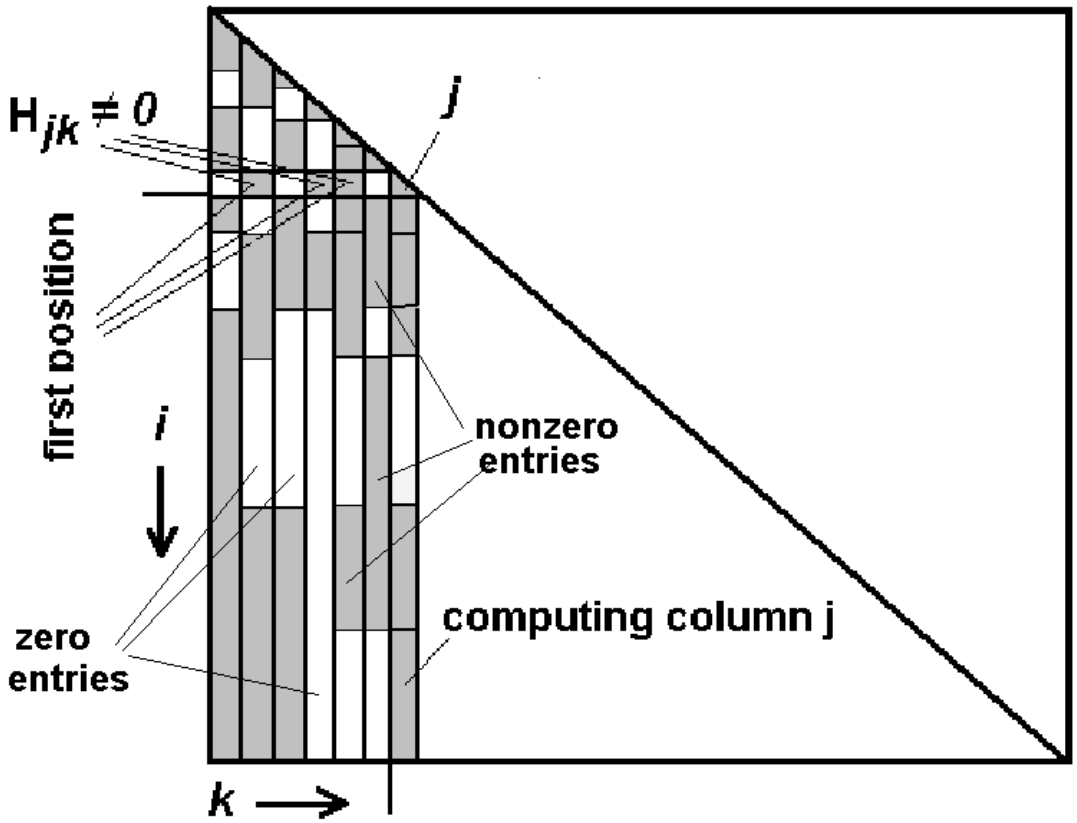
$$\mathbf{t}^j \leftarrow H_{j,2} \cdot \hat{\mathbf{H}}^2 + H_{j,4} \cdot \hat{\mathbf{H}}^4$$

1. Sparse technique



$$\mathbf{t}^j \leftarrow H_{j,2} \cdot \hat{\mathbf{H}}^2 + H_{j,4} \cdot \hat{\mathbf{H}}^4 + H_{j,5} \cdot \hat{\mathbf{H}}^5$$

1. Sparse technique



$$\mathbf{H}^j = \mathbf{H}^j - \mathbf{t}^j = \mathbf{H}^j - \sum_{H_{jk} \neq 0} H_{jk} \cdot \hat{\mathbf{H}}^k$$

Co daje technika sparse?

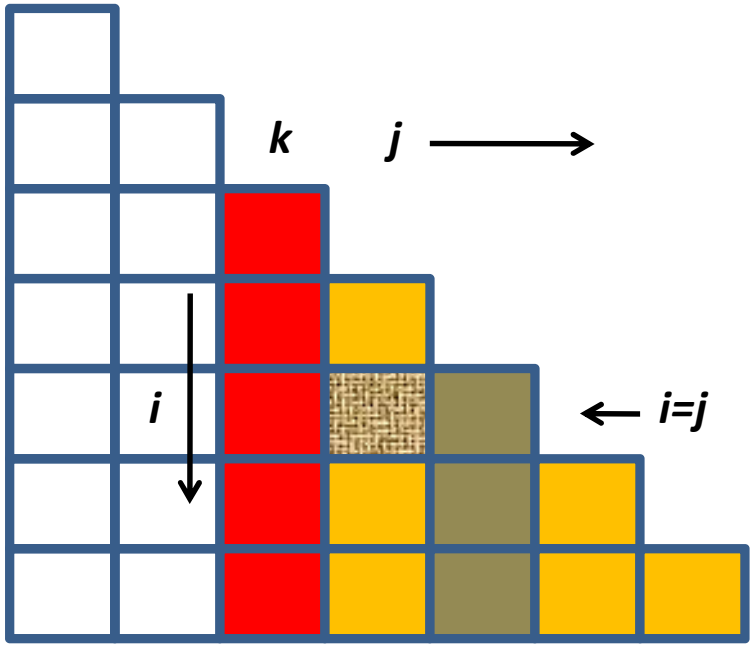
1. Przy obliczeniu wektora poprawek t_j pobieramy tylko kolumny, które mają niezerowy mnożnik l_{jk}.

2. Przy obliczeniu elementów wektora poprawek t_j w pętle po indeksu i uwzględniamy tylko niezerowe elementy kolumny s_k ($l_{ik} \neq 0$), przy czym $i<k$.

Powoduje to istotne zmniejszenie ilości operacji arytmetycznych w stosunku do innych technik, na przykład skyline.

Solwer right-looking sparse

Macierz gęsta



do k=1, N

- factorize column j

$l_{kk} = \sqrt{a_{kk}}$

do i=k+1,N

$l_{ik} = a_{ik}/l_{kk}$

end do

- update columns, located at right

do j = k+1, N

do i=j,N

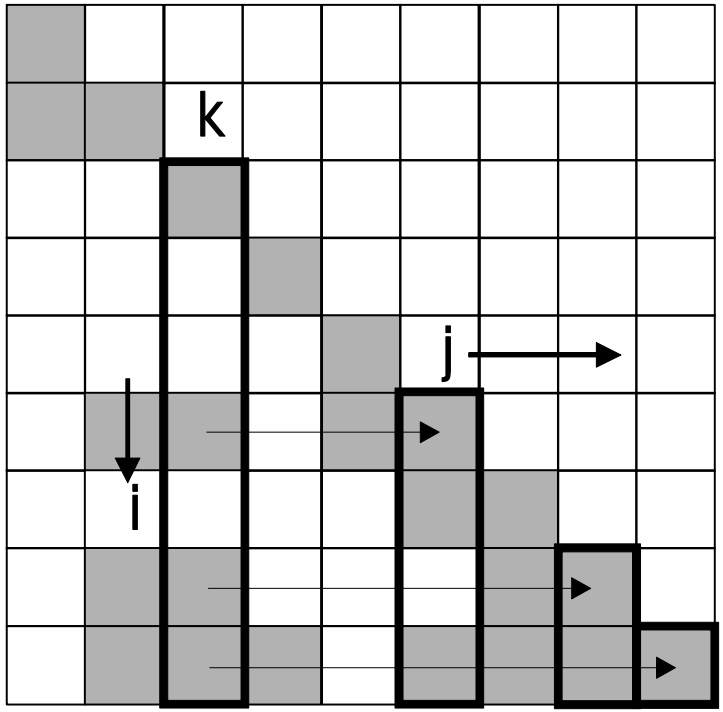
$a_{ij} = a_{ij} - l_{jk} \cdot l_{ik}$

end do

end do

end do

Macierz rzadka:



Obliczamy kolumnę *k*. **Kolumna** *k* **będzie poprawiała kolumny** $j = i$, **dla których elementy** $l_{ik} \neq 0$. **Poprawa kolumny** *j* **odbywa się tak:**

$$\tilde{a}_{ij} = a_{ij} - l_{ik} l_{jk}, \quad i \in F_k$$

F_k – **zbiór indeksów** *i*, **dla których** $l_{ik} \neq 0$

Solwer right-looking sparse

$$\begin{pmatrix} 10 & & & & \\ 0 & 10 & & & \\ 1 & 0 & 10 & & \\ 2 & 1 & 0 & 10 & \\ 0 & 1 & 2 & 1 & 10 \end{pmatrix} \rightarrow$$

k=1 (kolumna 1):

$l_{11}=\sqrt{a_{11}} = 3.162;$
$l_{i1}=a_{i1}/l_{11}$

$$\rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 10 & & & \\ 0.316 & 0 & 10 & & \\ 0.632 & 1 & 0 & 10 & \\ 0 & 1 & 2 & 1 & 10 \end{pmatrix}$$

Poprawiamy elementy kolumn, umieszczonych z prawa od kolumny j:

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 10 & & & \\ 0.316 & 0 & 10 & & \\ 0.632 & 1 & 0 & 10 & \\ 0 & 1 & 2 & 1 & 10 \end{pmatrix}$$

Liczymy macierz poprawek:

$$\begin{pmatrix} 0 \\ 0.316 \\ 0.632 \\ 0 \end{pmatrix} \cdot (0 \quad 0.316 \quad 0.632 \quad 0) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0.2 & 0 \\ 0 & 0.2 & 0.399 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

i poprawiamy:

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 10 & & & \\ 0.316 & 0 & 9.9 & & \\ 0.632 & 1 & -0.2 & 9.601 & \\ 0 & 1 & 2 & 1 & 10 \end{pmatrix}$$

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 10 & & & \\ 0.316 & 0 & 9.9 & & \\ 0.632 & 1 & -0.2 & 9.601 & \\ 0 & 1 & 2 & 1 & 10 \end{pmatrix} \rightarrow$$

k=2 (kolumna 2):

$l_{22}=\sqrt{a_{22}} = 3.162;$
$l_{i2}=a_{i2}/l_{22}$

$$\rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 9.9 & & \\ 0.632 & 0.316 & -0.2 & 9.601 & \\ 0 & 0.316 & 2 & 1 & 10 \end{pmatrix}$$

Poprawiamy kolumny 4, 5:

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 9.9 & & \\ 0.632 & 0.316 & -0.2 & 9.501 & \\ 0 & 0.316 & 2 & 0.9 & 9.9 \end{pmatrix}$$

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 9.9 & & \\ 0.632 & 0.316 & -0.2 & 9.501 & \\ 0 & 0.316 & 2 & 0.9 & 9.9 \end{pmatrix} \rightarrow$$

k=3 (kolumna 3):

$l_{33}=\sqrt{a_{33}} = 3.146;$
$l_{i3}=a_{i3}/l_{33}$

$$\rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 3.146 & & \\ 0.632 & 0.316 & -0.064 & 9.501 & \\ 0 & 0.316 & 0.636 & 0.9 & 9.9 \end{pmatrix}$$

Poprawiamy kolumny 5, 6:

$$\begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 3.146 & & \\ 0.632 & 0.316 & -0.064 & 9.497 & \\ 0 & 0.316 & 0.636 & 0.941 & 9.496 \end{pmatrix} \rightarrow$$

k=4 (kolumna 4):

$l_{44}=\sqrt{a_{44}} = 3.082;$
$l_{i4}=a_{i4}/l_{44}$

$$\rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 3.146 & & \\ 0.632 & 0.316 & -0.064 & 3.082 & \\ 0 & 0.316 & 0.636 & 0.305 & 9.403 \end{pmatrix} \rightarrow$$

k=5 (kolumna 5):

$l_{55}=\sqrt{a_{55}} = 3.066;$

$$\rightarrow \begin{pmatrix} 3.162 & & & & \\ 0 & 3.162 & & & \\ 0.316 & 0 & 3.146 & & \\ 0.632 & 0.316 & -0.064 & 3.082 & \\ 0 & 0.316 & 0.636 & 0.305 & 3.066 \end{pmatrix}$$

$for(k=1;k<=N;k++)$ // *obliczamy kolumnę* k

{

$a_{kk}=\sqrt{a_{kk}};$

// *obliczamy elementy kolumny* k, *umieszczone pod elementem* a_{kk}

$for(i\in L_k)$ // L_k – *zbiór indeksów* $i, i>k$

// *dla których* $a_{ik}\neq 0$

{

$$a_{ik}=\frac{a_{ik}}{a_{kk}}$$

}

// *poprawiamy elementy kolumn, umieszczonych*

// *sprawa od kolumny* k

$for(j\in \hat{L}_k)$ // $\hat{L}_k$ – *zbiór indeksów* $j\geq i$, *dla których* $a_{jk}\neq 0$

{

$for(i\in F_k)$ // F_k – *zbiór⋯wartosci indeksów* $i, i\geq j$

// *dla których* $a_{ik}\neq 0$

{

$a_{ij}=a_{ij}-a_{ik}a_{jk}$

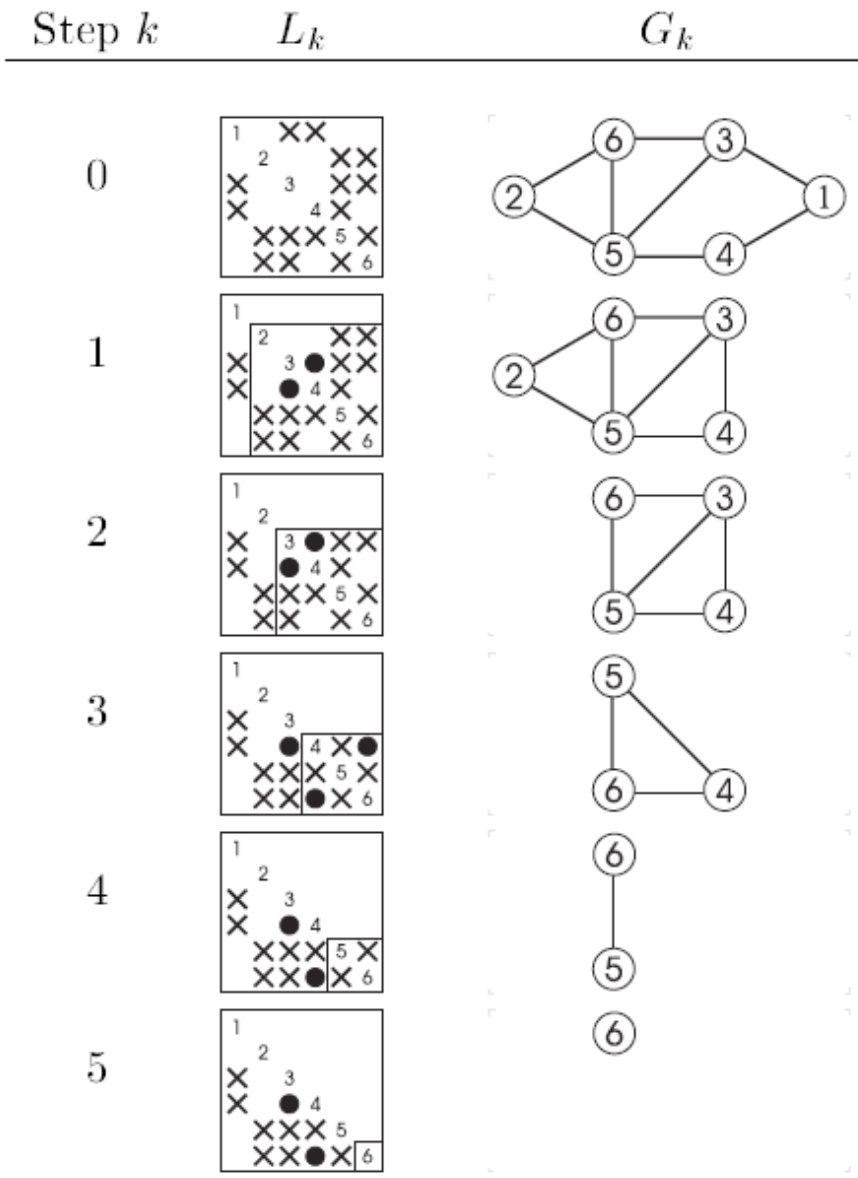
}

}

}

Solver right-looking sparse – algorytm ogólny

Solwer multifrontalny



Jest podana macierz rzadka. Po wykonaniu permutacji, odpowiadających wybranemu algorytmu uporządkowania, portret macierzy oraz graf przyległości są podane na step 0.

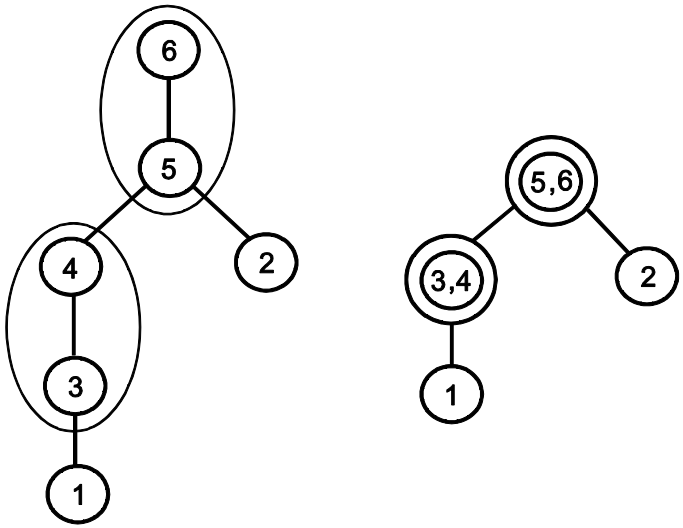
Dalej wykonana procedura faktoryzacji symbolicznej z celą wyjaśnienia, gdzie powstają zapełnienia.

Rysunek jest pobrany z [7], fig. 1.2

Dostajemy portret macierzy sfaktoryzowanej:

$$\begin{pmatrix} 1 & & x & x & & \\ & 2 & & & x & x \\ x & & 3 & \bullet & x & x \\ x & & \bullet & 4 & x & \bullet \\ & x & x & x & 5 & x \\ & x & x & \bullet & x & 6 \end{pmatrix}$$

Drzewo eliminacji:
a) - pierwotne
b) - superwęzłowe



a) b)

- **Analiza struktury tej macierzy polega na tworzeniu drzewa eliminacji.**

- **Wierzchołkami tego drzewa są kolumny macierzy.**

- **Rodzicem kolumny nazywają pierwszy niezerowy element, umieszczony pod diagonalą. Wskazuje na najbliższą kolumnę sprawa od podanej, która będzie poprawiona elementami podanej kolumny.**

- **Na przykład, kolumna 1 będzie modyfikowała kolumnę 3 – numer wiersza pierwszego niezerowego elementu w kolumnie 1 jest równy 3. Kolumna 2 będzie modyfikowała elementy kolumny 5, kolumna 3 – elementy 4 i t. d. Zależność ta jest przedstawiona w postaci pierwotnego drzewa eliminacji.**

- **Faktoryzacje macierzy można zaczynać od kolumny 1 albo od kolumny 2 albo równolegle eliminować kolumny 1 i 2 (dla tego że pomiędzy nimi nie istnieje żadnej zależności).**

- **Kluczowym faktorem podniesienia wydajności jest możliwość łączenia kolumn macierzy w grupy – nadaje możliwość zastosowania faktoryzacji blokowej. Nie każde sąsiednie kolumny można łączyć w grupę (blok). Zależy to od struktury macierzy rzadkiej.**

- **Superwęzłem nazywamy łączenie kilkach wierzchołków drzewa eliminacji, które mają sekwencyjną numeracje, tworzą jedną ścieżkę i mają ten samy wierzchołek ze starszym numerem. W macierzy rzadkiej każdemu superwęzłowi odpowiada gęsta trójkątna podmacierz, umieszczona bezpośrednio pod diagonalą. Kolumny, które odpowiadają superwęzłowi, będą łączone w bloki.**

k	Asemblowanie	F_k	U_k	C_k	L_k
1	–	$\begin{array}{c} \quad 1 \quad 3 \quad 4 \\ 1 \\ 3 \\ 4 \end{array}\begin{pmatrix} 1 & x & x \\ x & & \\ x & & \end{pmatrix}$	$\begin{array}{c} \quad 3 \quad 4 \\ 3 \\ 4 \end{array}\begin{pmatrix} 3 & \# \\ \# & 4 \end{pmatrix}$	$\begin{array}{c} \quad 1 \\ 1 \\ 3 \\ 4 \end{array}\begin{pmatrix} 1 \\ x \\ x \end{pmatrix}$	$\begin{pmatrix} 1 & & \\ & & \\ x & & \\ x & & \end{pmatrix}$
2	–	$\begin{array}{c} \quad 2 \quad 5 \quad 6 \\ 2 \\ 5 \\ 6 \end{array}\begin{pmatrix} 2 & x & x \\ x & & \\ x & & \end{pmatrix}$	$\begin{array}{c} \quad 5 \quad 6 \\ 5 \\ 6 \end{array}\begin{pmatrix} 5 & \# \\ \# & 6 \end{pmatrix}$	$\begin{array}{c} \quad 2 \\ 2 \\ 5 \\ 6 \end{array}\begin{pmatrix} 2 \\ x \\ x \end{pmatrix}$	$\begin{pmatrix} 1 & & \\ & 2 & \\ x & & \\ x & & \\ & x & \\ & x & \end{pmatrix}$
3	$+U_1$	$\begin{array}{c} \quad 3 \quad 4 \quad 5 \quad 6 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{pmatrix} 3 & \# & x & x \\ \# & 4 & x & \\ x & x & & \\ x & & & \end{pmatrix}$	$\begin{array}{c} \quad 5 \quad 6 \\ 5 \\ 6 \end{array}\begin{pmatrix} 5 & \# \\ \# & 6 \end{pmatrix}$	$\begin{array}{c} \quad 3 \quad 4 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{pmatrix} 3 & \\ \# & 4 \\ x & x \\ x & \# \end{pmatrix}$	$\begin{pmatrix} 1 & & & \\ & 2 & & \\ x & & 3 & \\ x & & \# & 4 \\ & x & x & x \\ & x & x & \# \end{pmatrix}$
4	$+U_2+U_3$	$\begin{array}{c} \quad 5 \quad 6 \\ 5 \\ 6 \end{array}\begin{pmatrix} 5 & x \\ x & 6 \end{pmatrix}$	–	$\begin{array}{c} \quad 5 \quad 6 \\ 5 \\ 6 \end{array}\begin{pmatrix} 5 & \\ x & 6 \end{pmatrix}$	$\begin{pmatrix} 1 & & & & \\ & 2 & & & \\ x & & 3 & & \\ x & & \# & 4 & \\ & x & x & x & 5 \\ & x & x & \# & x & 6 \end{pmatrix}$

- **Proces eliminacji odbywa się przez cały czas w szeregu macierzy gęstych – frontalnych macierzy F.**

- **Macierzy U przechowują poprawki do frontalnych macierzy na następnych krokach faktoryzacji.**

- **Macierzy C zawierają części wyników ostatecznych – kompletnie faktoryzowanę podmacierzy.**

Blokowa multifrontalna metoda podkonstrukcji

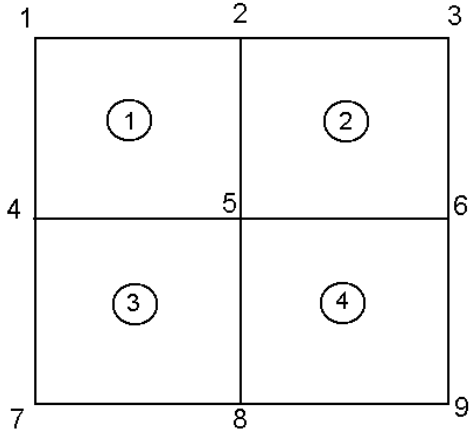
- **dowolna metoda uporządkowania**

- **Uporządkowanie grafu przyległości dla węzłów modelu MES, a nie dla równań:**
  - **znacznie mniej objętość danych**
  - **szybciej działa metoda uporządkowania**
  - **zwykłe mniej elementów niezerowych ponieważ graf spójności jest zgrubiony w porównaniu do grafa dla równań**
  - **powstaje blokowanie równań naturalne, a nie na podstawie analizy macierzy – lepiej jest skupienie do większych bloków**

- **Eliminacje węzeł – po – węźle - na każdym kroku eliminacji eliminujemy równania, skojarzone z danym węzłem.**

- **Front – obiekt klasy C++, który zawiera:**
  - **numer eliminowanego węzła**
  - **listę węzłów, tworzących front**
  - **listę frontów poprzednich – numery frontów, które tworzą dany front**
  - **listę elementów skończonych, które tworzą dany front**

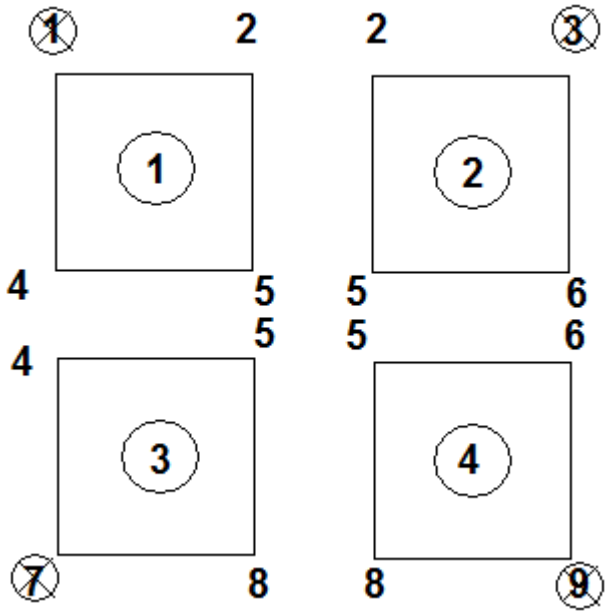
Przykład: płyta z siatką 2x2



№ węzła do wyeliminowania	Lista elementów, zawierających podany węzeł
1	1
3	2
7	3
9	4
2	1,2
6	2,4
8	3,4
4	1,3
5	1,2,3,4

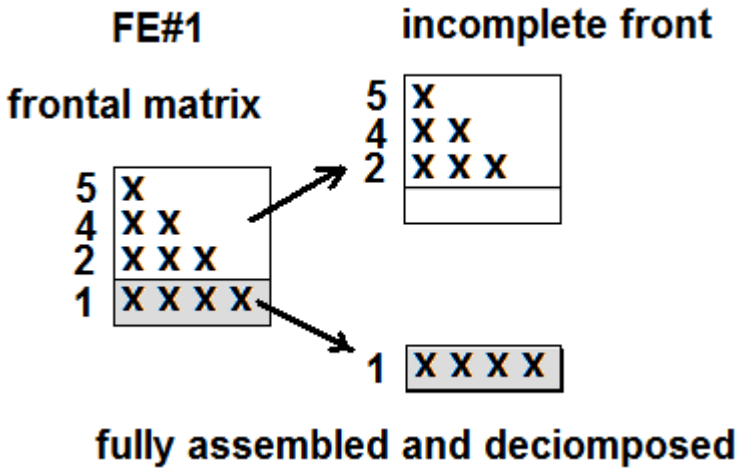
MMD nadaje taką kolejność eliminowania węzłów: 1,3,7,9,2,6,8,4,5.

1. **Rozdzielamy model FEM na oddzielne elementy skończone**
2. **Uzyskujemy taką kolejność dostarczania elementów skończonych, żeby spełnić podaną przez renumerator kolejność wyeliminowania węzłów.**
3. **Kompletnie zebrane węzły – dostarczanie pozostałych elementów skończonych nie zmienia współczynników tych równań.**
4. **Kompletnie zebrane równania muszą być wyeliminowane na danym kroku eliminacji**

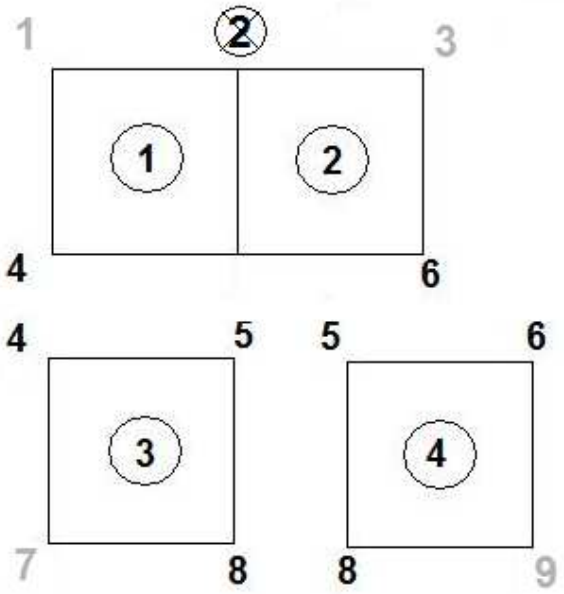


Eliminujemy węzły 1, 3, 7, 9

Macierz frontalna dla wyeliminowania węzła 1. Dla uproszczenia porozumienia będziemy uważali, że każdy węzeł zawiera tylko jedno równanie. Węzły 3, 7, 9 będą wyeliminowane tak samo, jak i węzeł 1.

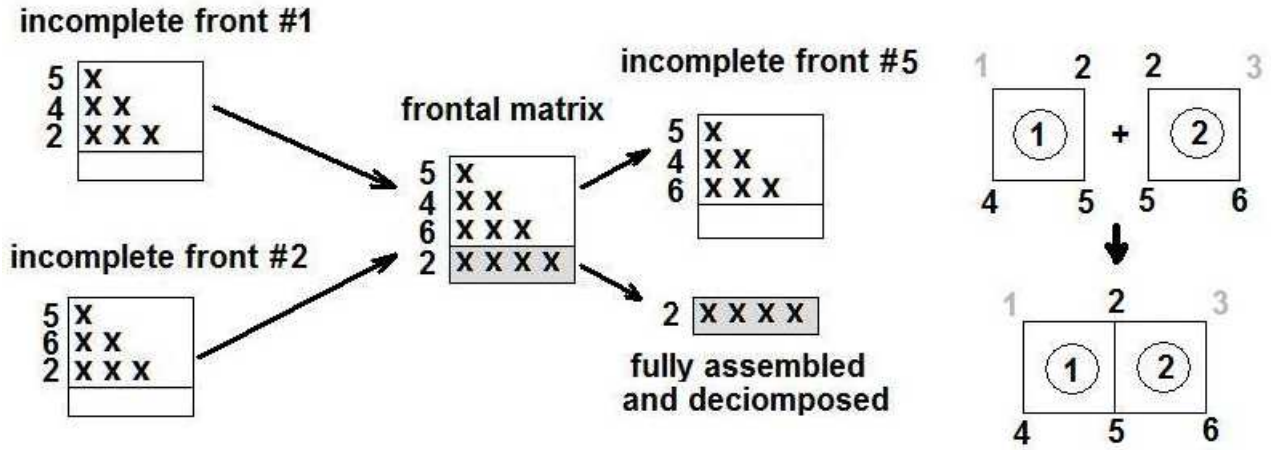


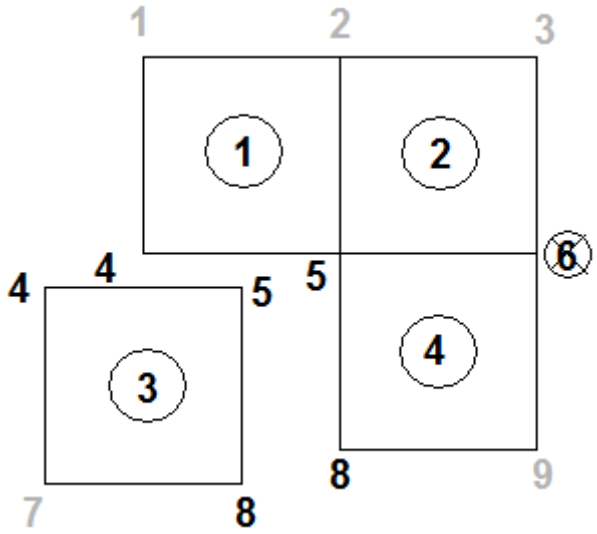
W macierzy frontalnej umieszczamy węzły w kolejności odwrotnej do podanej przez renumerator – kompletnie zebrane równania są w dolnej części



Eliminujemy węzeł 2

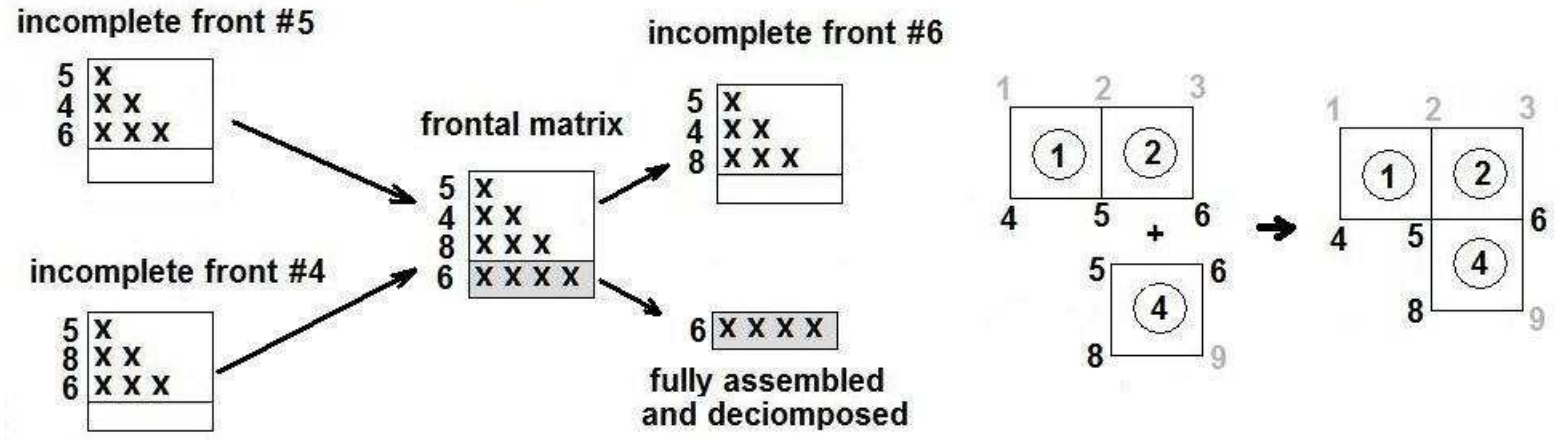
Macierz frontalna dla eliminowania węzła 2 pozostaje zebrana z frontów niezakończonych 1, 2. Na kroku podanym nowe elementy skończone już nie będą dostarczane.

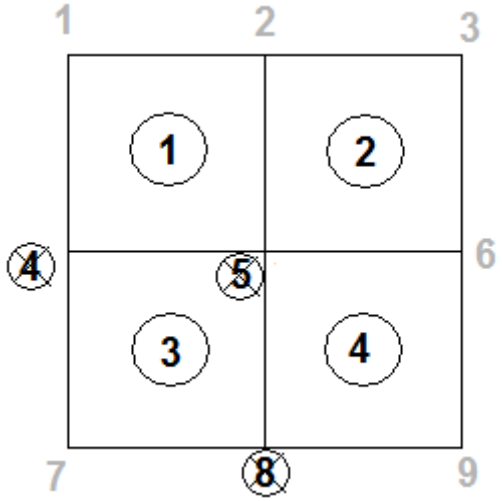




Eliminujemy węzeł 6

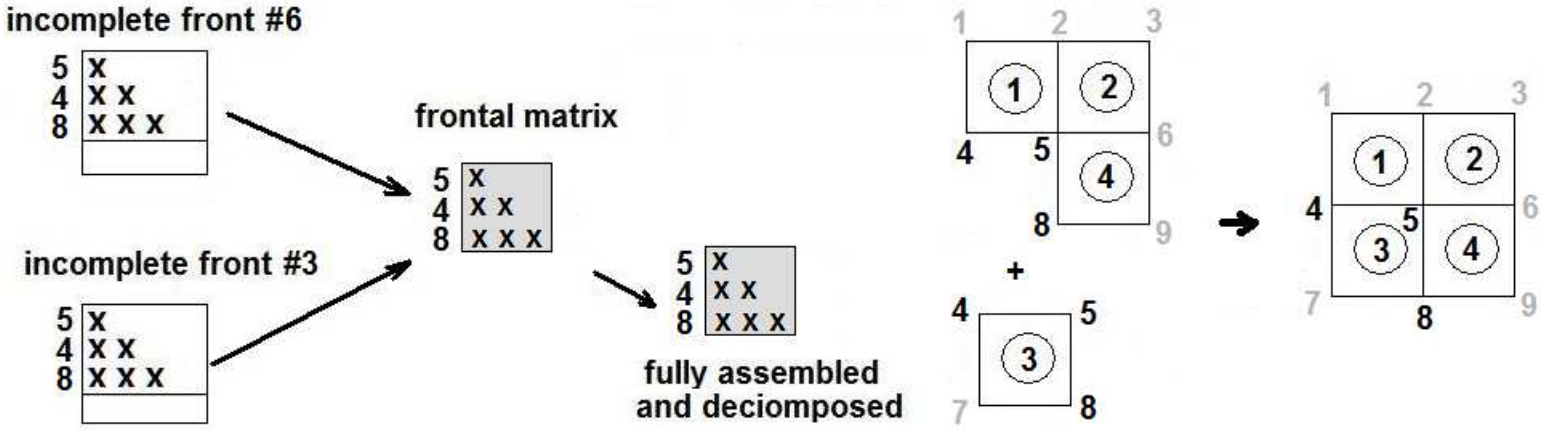
Macierz frontalna dla eliminowania węzła 6 pozostaje zebrana z frontów niezakończonych 4, 5. Na kroku podanym nowe elementy skończone już nie będą dostarczane.





Eliminujemy węzły 8, 4, 5

Macierz frontalna dla eliminowania węzłów 8, 4, 5 pozostaje zebrana z frontów niezakończonych 6, 3. Na kroku podanym nowe elementy skończone już nie będą dostarczane.



Struktura danych „Deskryptor procesu”:

Elimination step	Eliminated node	List of frontal nodes	List of previous fronts	List of assembling FE
1	1	5, 4, 2, 1	-----	1
2	3	5, 6, 2, 3	-----	2
3	7	5, 4, 8 , 7	-----	3
4	9	5, 8, 6, 9	-----	4
5	2	5, 4, 6, 2	1,2	----
6	6	5, 4, 8, 6	5,4	----
7	8	5, 4, 8	6,3	----
8	4	5, 4	7	----
9	5	5	8	----

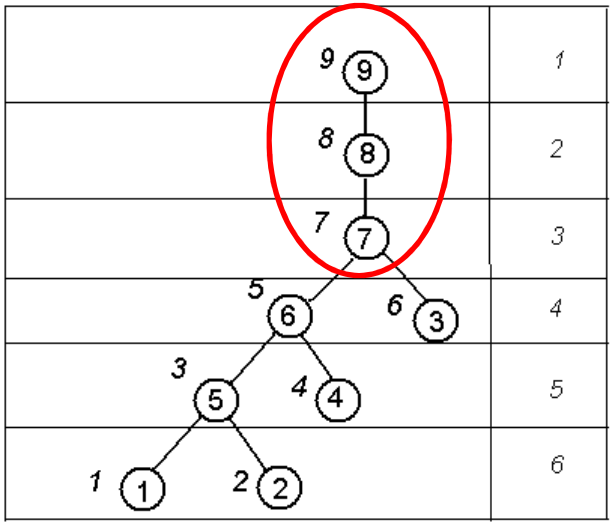
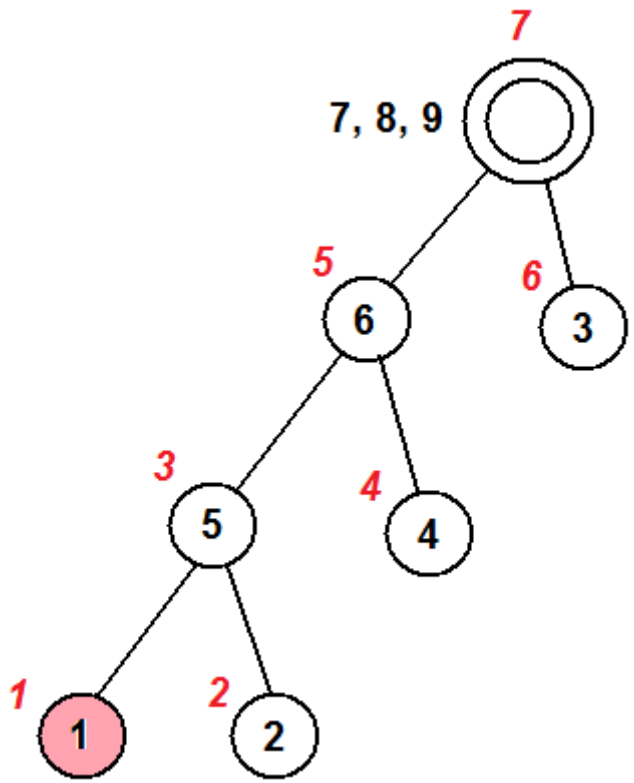


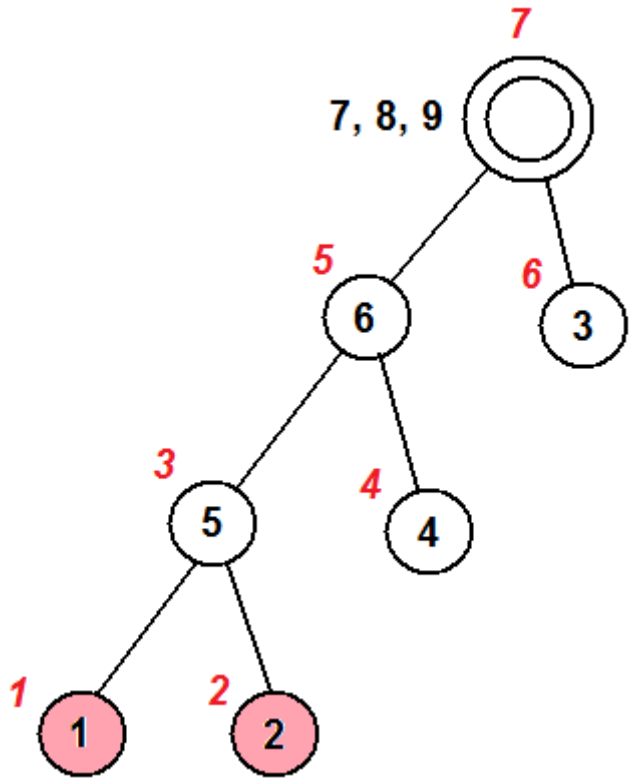
Рис. 2. Struktura poziomów drzewa frontów

- **Uporządkowanie drzewa frontalnego z celą zmniejszenia objętości danych dla przechowywania niezakończonych frontów**
- **Łączenie frontów sekwencyjnych powoduje zwiększenie rozmiaru bloku równań kompletnie zebranych – klucz do podniesienia wydajności przy faktoryzacji. Optymalny rozmiar bloku: M = 60 – 80 .**

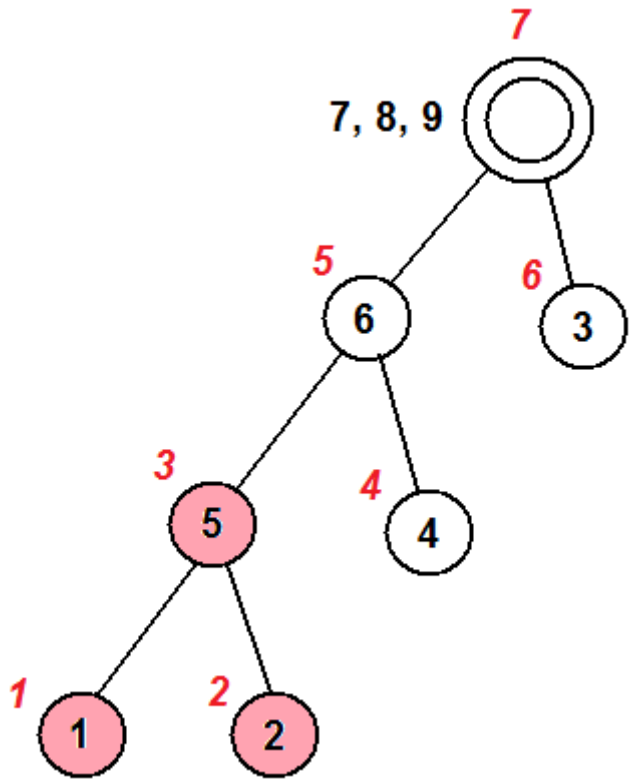
➢ **Proces eliminowania jest wykonany krok po kroku zgodnie kolejności eliminowania węzłów**



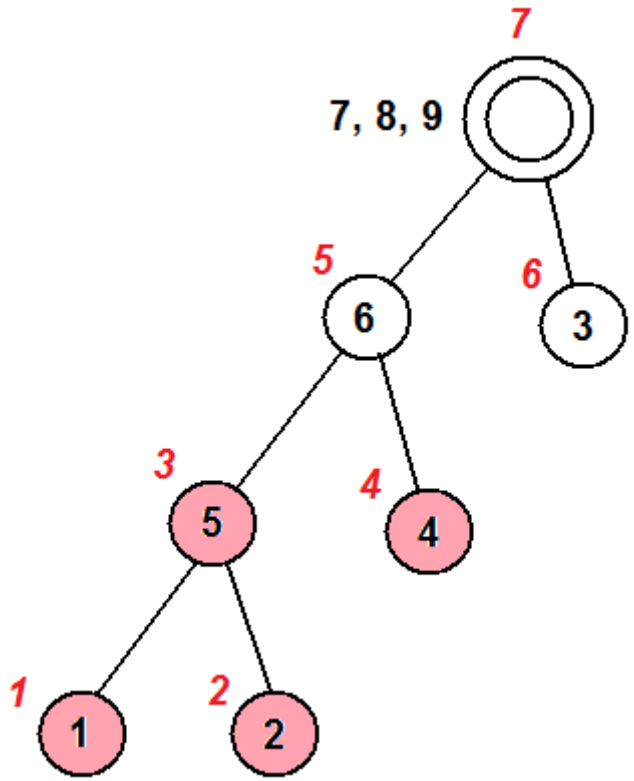
- **Proces eliminowania jest wykonany krok po kroku zgodnie kolejności eliminowania węzłów**



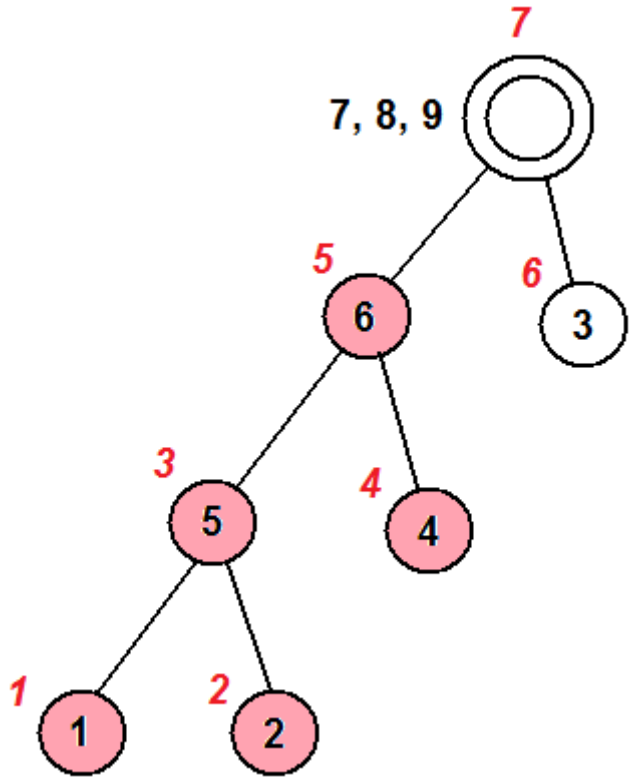
- **Proces eliminowania jest wykonany krok po kroku zgodnie kolejności eliminowania węzłów**



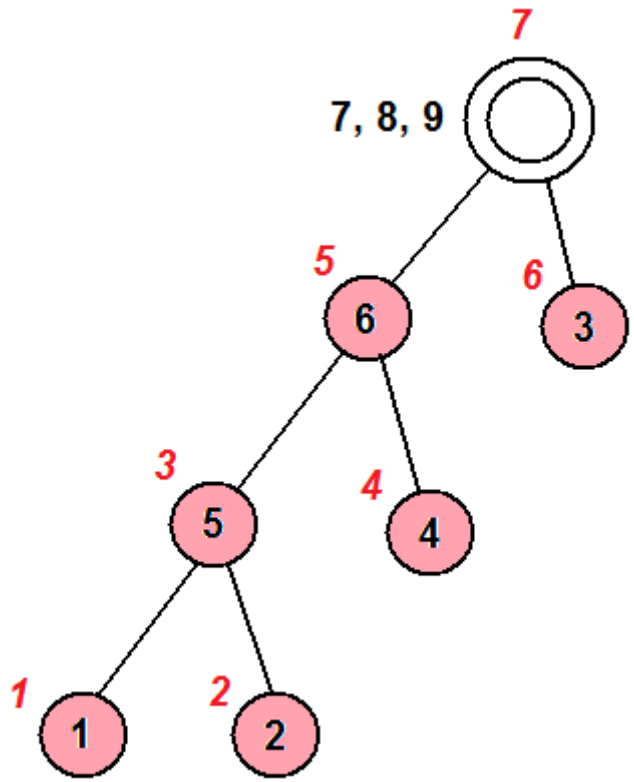
➢ **Proces eliminowania jest wykonany krok po kroku zgodnie kolejności eliminowania węzłów**



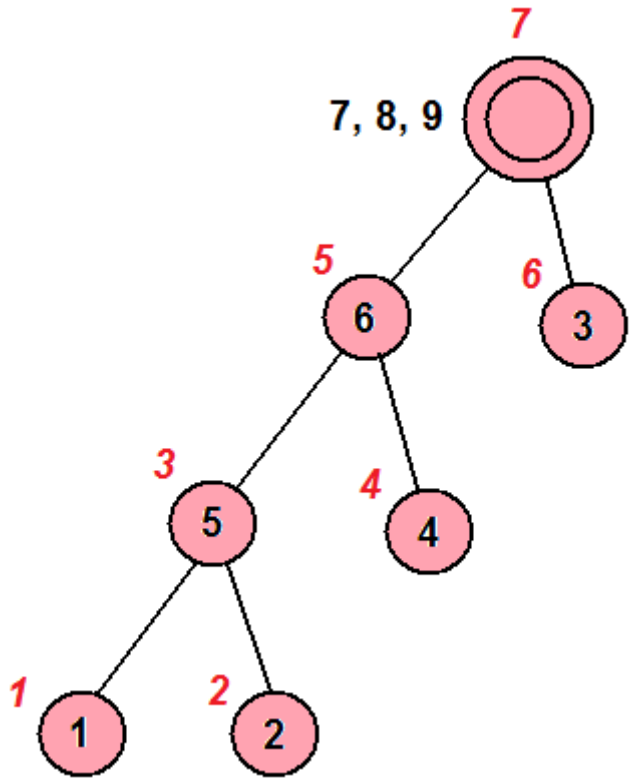
➢ **Proces eliminowania jest wykonany krok po kroku zgodnie kolejności eliminowania węzłów**



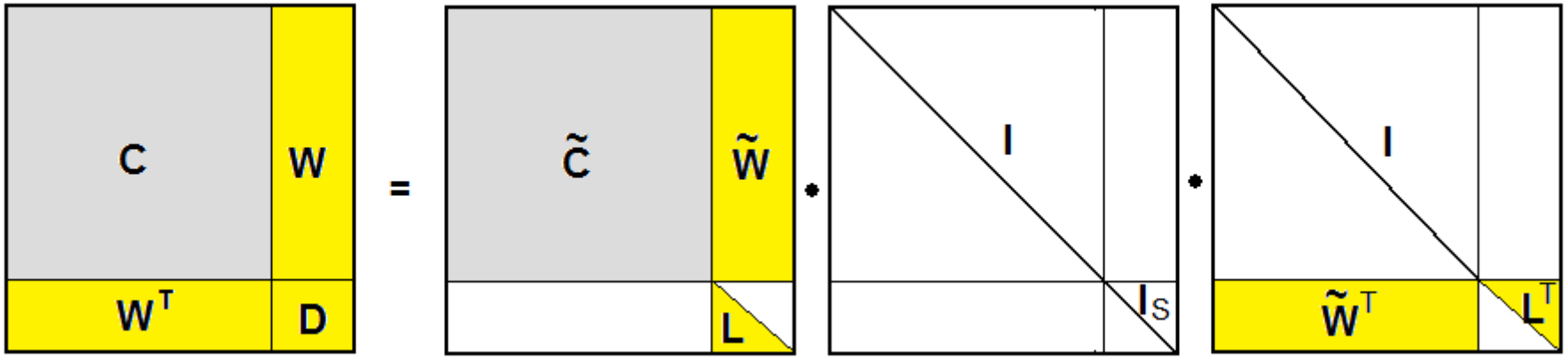
- **Proces eliminowania jest wykonany krok po kroku zgodnie kolejności eliminowania węzłów**



- **Proces eliminowania jest wykonany krok po kroku zgodnie kolejności eliminowania węzłów**



Blokowa faktoryzacja Choleskiego w macierzy frontalnej

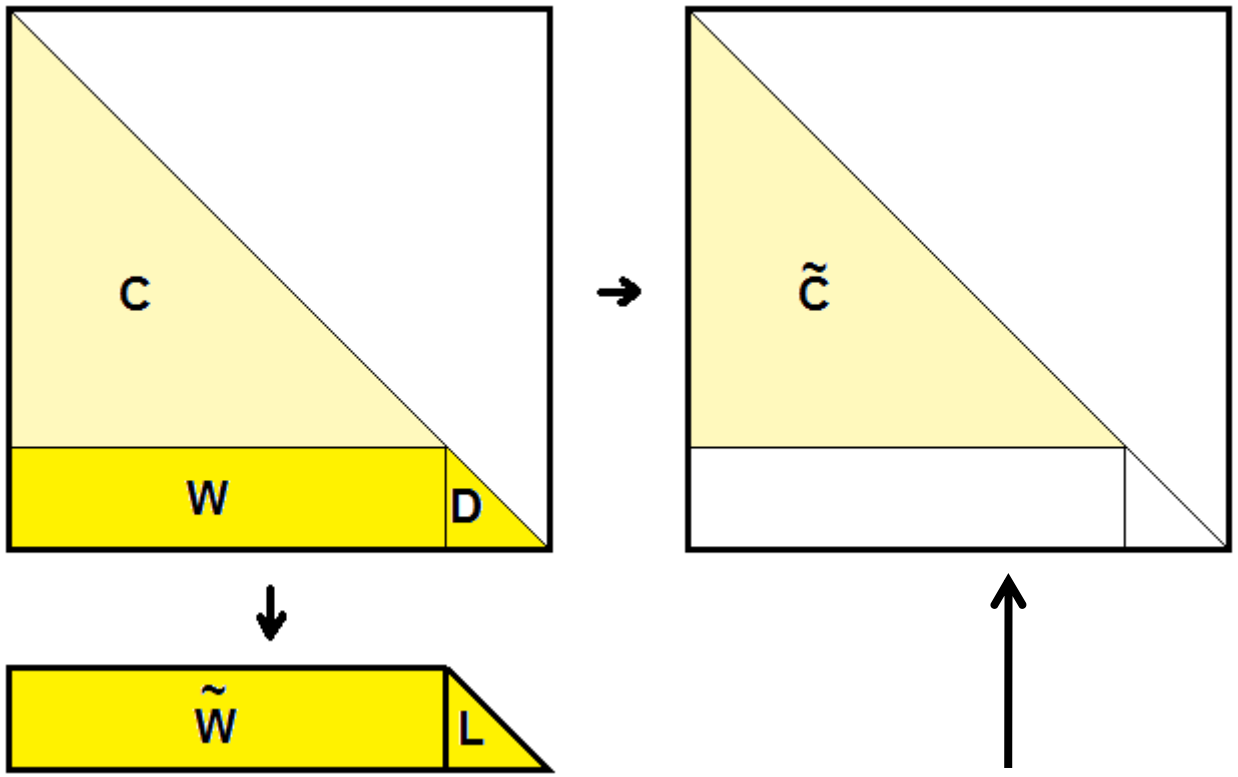


$$\begin{pmatrix} \mathbf{C} & \mathbf{W} \\ \mathbf{W}^T & \mathbf{D} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{C}} & \tilde{\mathbf{W}} \\ 0 & \mathbf{L} \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{I}_S \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ \tilde{\mathbf{W}}^T & \mathbf{L}^T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

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

Symetryczny schemat przechowywania:

Ponieważ kolejność lokalnego numerowania węzłów jest odwrotną do numerowania globalnego Perm, blok kompletnie zebranych równań jest umieszczony w dolnej części macierzy – unikamy wolnej procedury kopiowania elementów bloku C przy modyfikowaniu



Bufor dla kompletnie zebranej i sfaktoryzowanej części macierzy

Niezakończony front

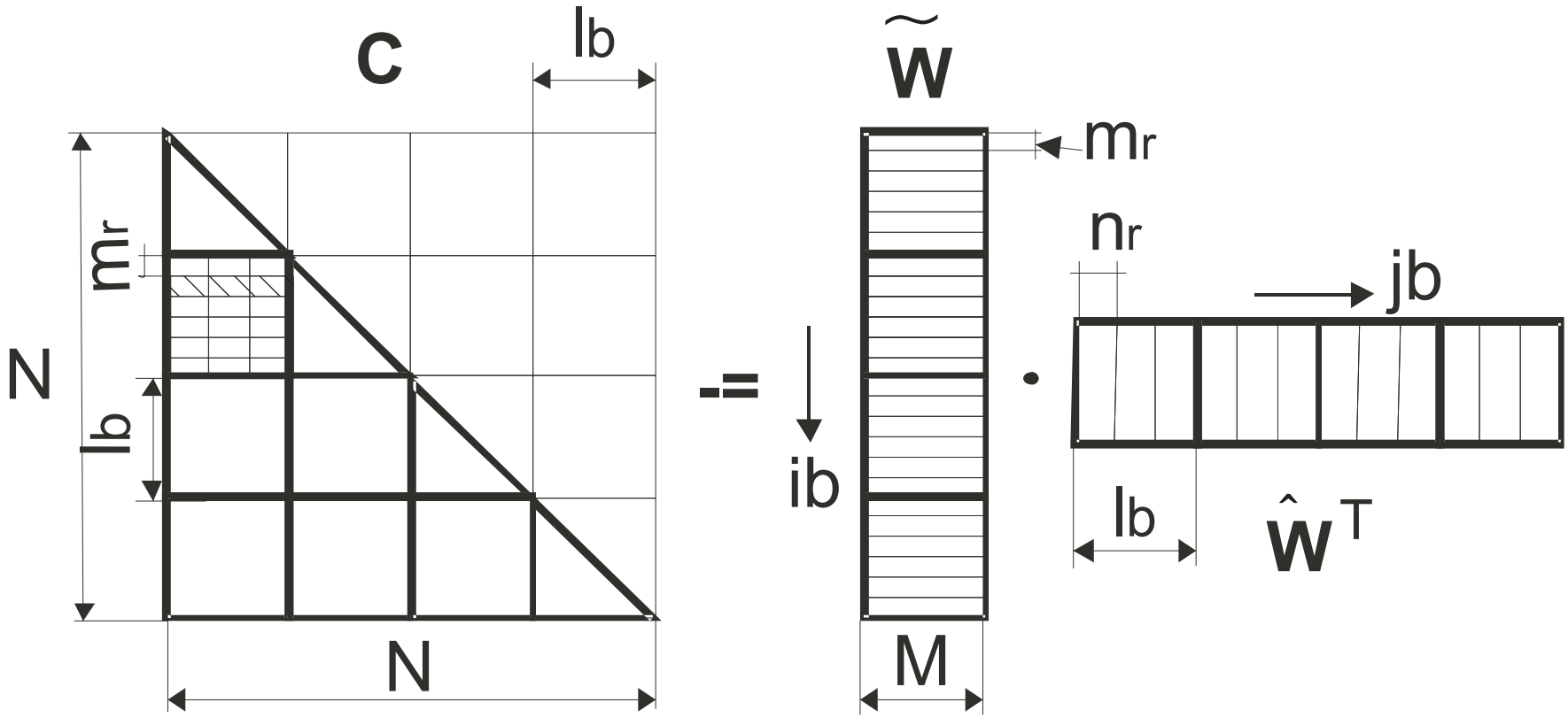
Faktoryzacja macierzy frontalnej:

$$\tilde{\mathbf{C}} = \mathbf{C} - \tilde{\mathbf{W}} \cdot \underbrace{\mathbf{I}_S \cdot \tilde{\mathbf{W}}^T}_{\hat{\mathbf{W}}^T}$$

```
// 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
    }
}
```

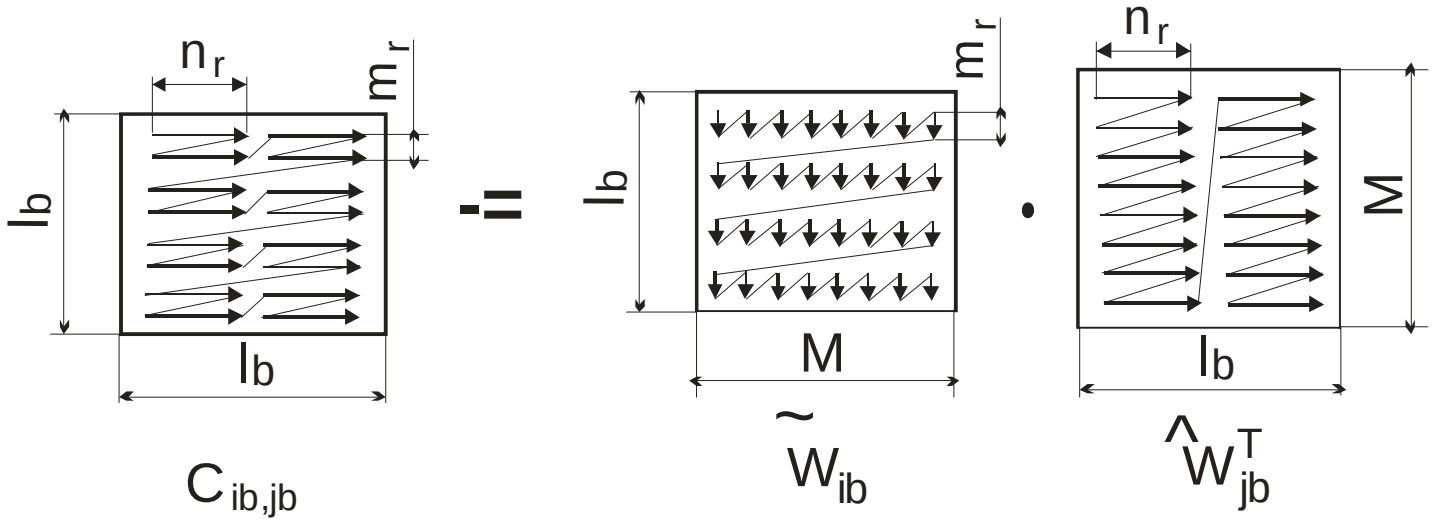
Symetryczny schemat przechowywania wymaga utrzymywać w pamięci RAM dla każdej macierzy frontalnej ~$N^2/2$ elementów zamiast N^2 . Jednak uniemożliwia to zastosowanie procedur wysokiej wydajności DGEMM, DSYRK z bibliotek BLAS, Intel MKL, IMSL,…, którzy wymagają ogólnego schematu przechowywania nawet dla macierzy symetrycznych. Rozwiązanie: stworzyć swoi własne procedury wysokiej wydajności.

Cache blocking and register's blocking:

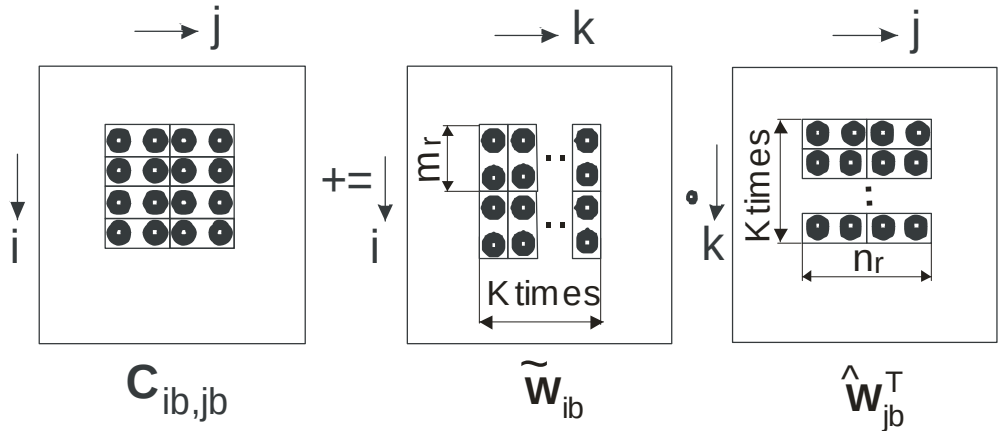


Dzielimy macierzy na bloki:
Grube linie – cache blocking, cienkie – register's blocking $m_r \times n_r$

Pakowanie danych dla bloków macierzy z celą minimalizowania cache misses:



Blokowanie of XMM rejestrów (platforma Intel 64) i rozwijanie petli wewnętrznej:



Wyniki:

Komputer: Intel® Core™2 Quad CPU Q6600 @2.40 GHz, RAM DDR2 800 MHz 8 GB

1. **Test dla macierzy gęstej (N = 4 800, rozmiar bloku kompletnie zebranych równań M = 48). Cel: oszacowanie wydajności i skalowalności faktoryzacji macierzy frontalnej**

Performance of factoring of frontal matrix (blocking of XMM registers, cache blocking, pack of data…)

Platform	Ia32				Intel 64 (x64)			
Number of processors	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

Performance of factoring of frontal matrix (cache blocking only)

Platform	ia32				Intel 64 (x64)			
Number of processors	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

2. Płyta kwadratowa z siatką 400×400 (964 794 równań)

Numerical factoring of the stiffness matrix, s

ANSYS 11.0 – multifrontalna metoda klasyczna

BSMFM: Blokowa multifrontalna metoda podkonstrukcji

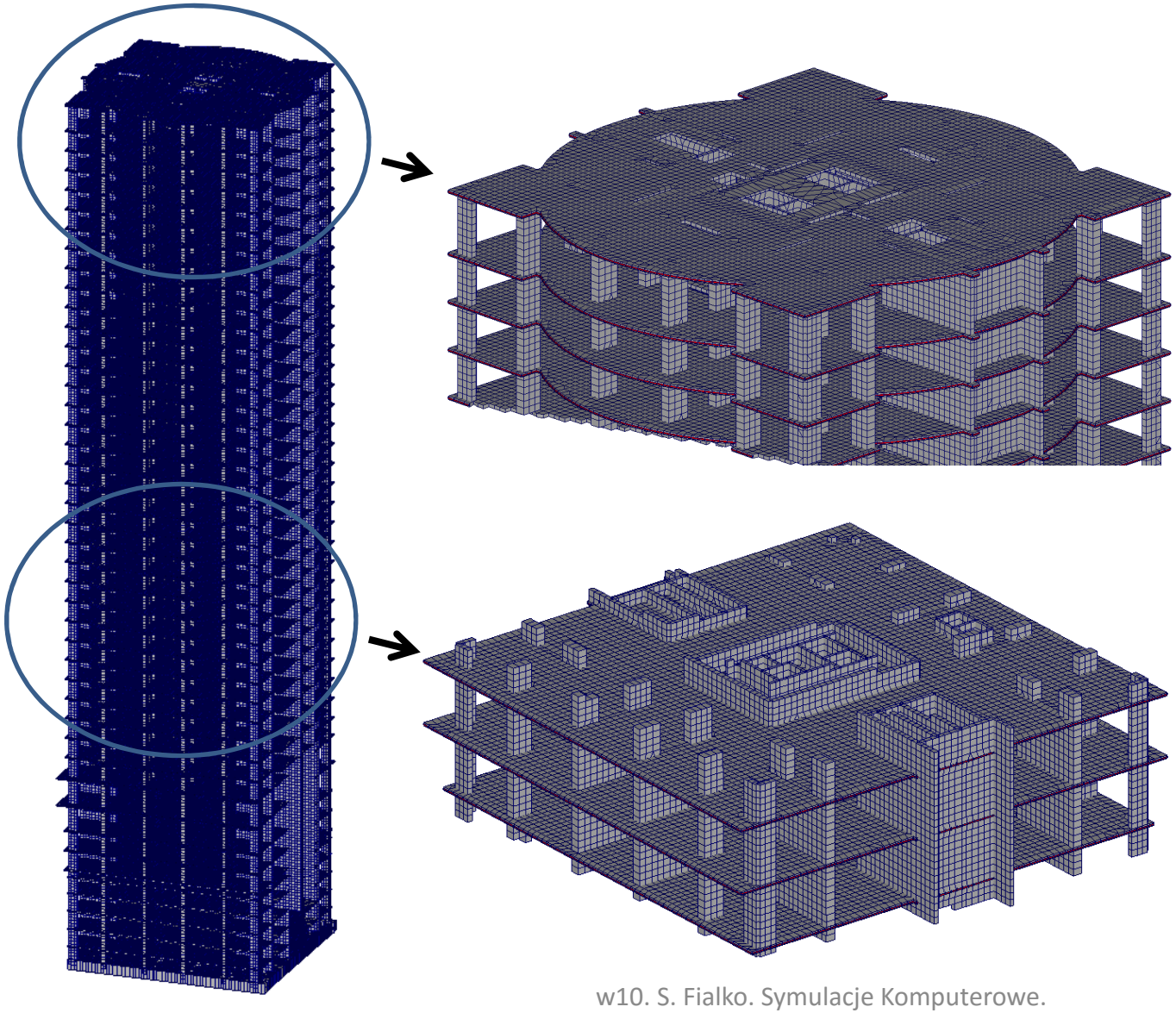
Number of processors	Platform ia32		Platform Intel 64
	ANSYS 11.0	BSMFM	BSMFM
1	221	117	86
2	176	90	60
4	159	78	51

3. Płyta kwadratowa z siatką 800×800 (3 849 594 równań)

Numerical factoring of the stiffness matrix, s

Number of processors	Platform ia32		Platform Intel 64
	ANSYS 11.0	BSMFM	BSMFM
1	Insufficient RAM	978	888
2	Insufficient RAM	768	551
4	Insufficient RAM	670	460

4. Model MES budynku wielopiętrowego (1 956 634 równań)



PARDISO (Intel MKL):

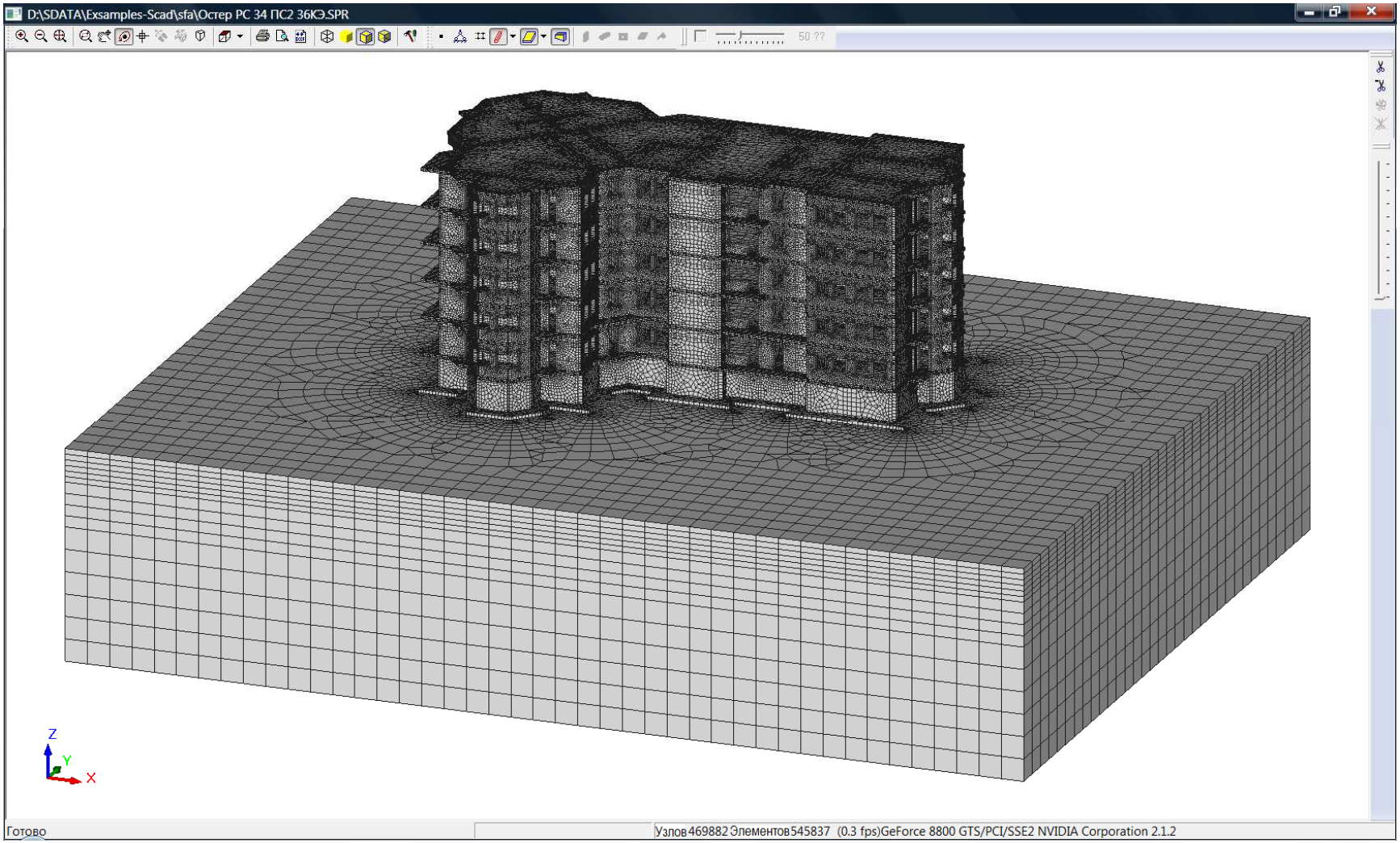
Insufficient RAM (8 GB) on platforms ia32, Intel 64

BSMFM on 4 processors:

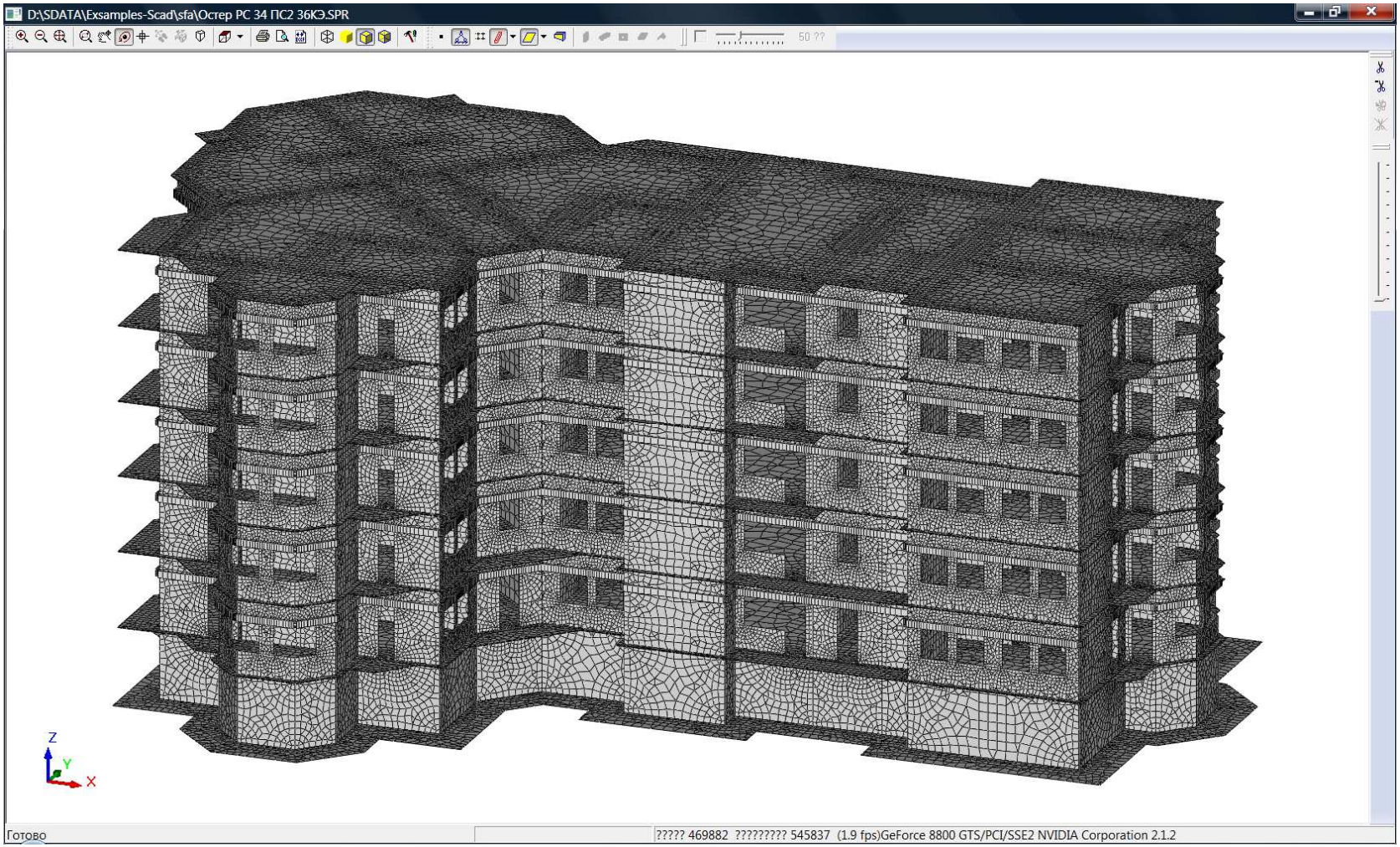
Ia32 : 443 s
Intel 64: 413 s

Size of factored stiffness matrix: 7.2 GB

5. Zadanie interakcji grunt-budynek (2 763 181 równań), rozmiar macierzy sfaktoryzowanej - 14 GB



BSMFM, platform Intel 64, 4 processors: duration of numerical factoring is 1157 s (19 m 17 s)



Underground part of structure

Rozwinięcie blokowej multifrontalnej metody podkonstrukcji

v 2002 : wersja nie blokowa, LU faktoryzacja macierzy frontalnej wierz – po - wierszę (Robot Millennium , SCAD v. 31)

v 2004 – 2008: cache blocking only, brakuje łączenie bloków kompletnie zebranych równań, LSL^T faktoryzacja macierzy frontalnej (SCAD v. 31, v 11)

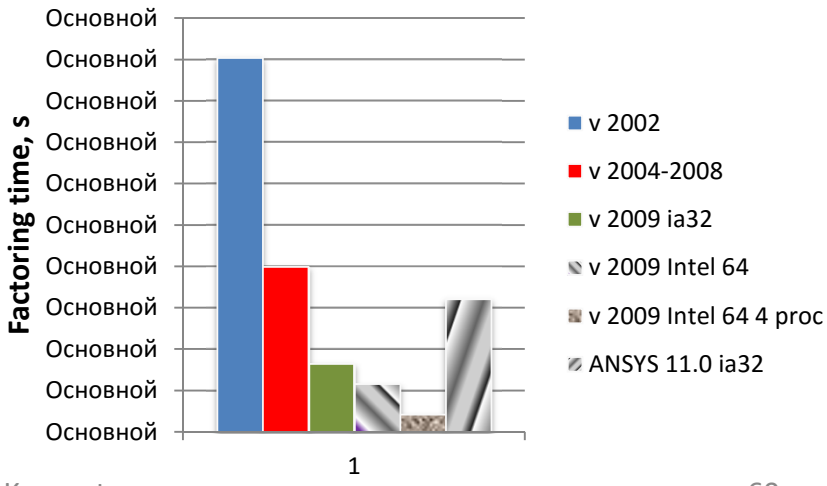
v 2009 : wektoryzowanie, XMM register's blocking, cache blocking, pakowanie danych, wielowątkowość (SCAD v. 11.3)

Test: cube 50×50×50, objętościowe elementy skończone, 397 941 równań

Duration of numerical factoring, s

Version, year	Platform	Number of processors		
		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

Duration of numerical factoring on single processor, s



References

1. Duff I. S., Reid J. K. The multifrontal solution of indefinite sparse symmetric linear systems, ACM Transactions on Mathematical Software, 9, pp. 302–325, 1983

2. Fialko S. Yu. Stress-Strain Analysis of Thin-Walled Shells with Massive Ribs, Int. App. Mech., 40, N4, pp. 432–439, 2004

3. Fialko S. Yu. The block substructure multifrontal method for solution of large finite element equation SETS. Technical Transactions, 1-NP/2009, issue 8, p.175 – 188.

4. Fialko S. Yu. A block sparse shared-memory multifrontal finite element solver for problems of structural mechanics. Computer Assisted Mechanics and Engineering Sciences, 16, 2009. p. 117 – 131.

5. Fialko S. Yu. PARFES: A method for solving finite element linear equations on multi-core computers. Advances in Engineering software. v 40, 12, 2010, pp. 1256 – 1265.

5. Goto K., Van De Geijn R. A. Anatomy of High-Performance Matrix Multiplication, ACM Transactions on Mathematical Software, V. 34, 3, pp. 1–25, 2008.

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

References

7. Dobrian F., Kumfert G., Pothen A., 2000. The Design of Sparse Direct Solvers using Object-Oriented Techniques, In: Bruaset A M, Langtangen H P, Quak E (eds.) Modern Software Tools in Scientific Computing. Springer-Verlag, pp 89–131.