Metody supernodalne jako alternatywa metodom wielofrontalnym

- **Zalety.** **Metoda wielofrontalna oszczędnie pracuje z pamięcią główną, nadaje się do wirtualizacji – jeśli rozmiar zadania przekracza możliwości pamięci głównej, dane, przeznaczone dla przechowywania macierzy poprawek, mogą być zapisane na dysk. Oprócz tego, bloki, zawierające części macierzy sfaktoryzowanej, też mogą być wyładowane na dysk.**

- **Wady.** **Metoda wielofrontalna wykonuje dużo transferów danych z jednego obszaru pamięci do innego, nawet jeśli zadanie mieści się w pamięci głównej. Występuje to przy kopiowaniu kompletnie sfaktoryzowanej części macierzy frontalnej, przy agregowaniu kolejnej macierzy frontalnej itd.**

Metody, alternatywne metodzie wielofrontalnej

- **Takie operacji należą do procedur niskiej wydajności, oprócz tego dla komputerów z pamięcią wspólną procedury kopiowania są przyspieszone bardzo słabo przy zwiększeniu ilości wątków.**

- **Obniża to wydajność solwerów wielofrontalnych oraz ich speed up.**

- **Powstaje interes do rozwinięcia metod alternatywnych, orientowanych w pierwszej kolejce na komputery wielordzeniowe.**

PARDISO

(Intel Math Kernel Library)

- **Ograniczymy się rozważaniem solwera PARDISO dla macierzy symetrycznych, mających niezerowe elementy diagonalne. To są typowe macierzy sztywności MES dla zadań mechaniki ciała sztywnego oraz mechaniki konstrukcji. Takie macierzy nie wymagają wykonania permutacji w trakcie faktoryzacji numerycznej, ponieważ na każdym kroku wiodącym elementem służy odpowiedni element diagonalny.**

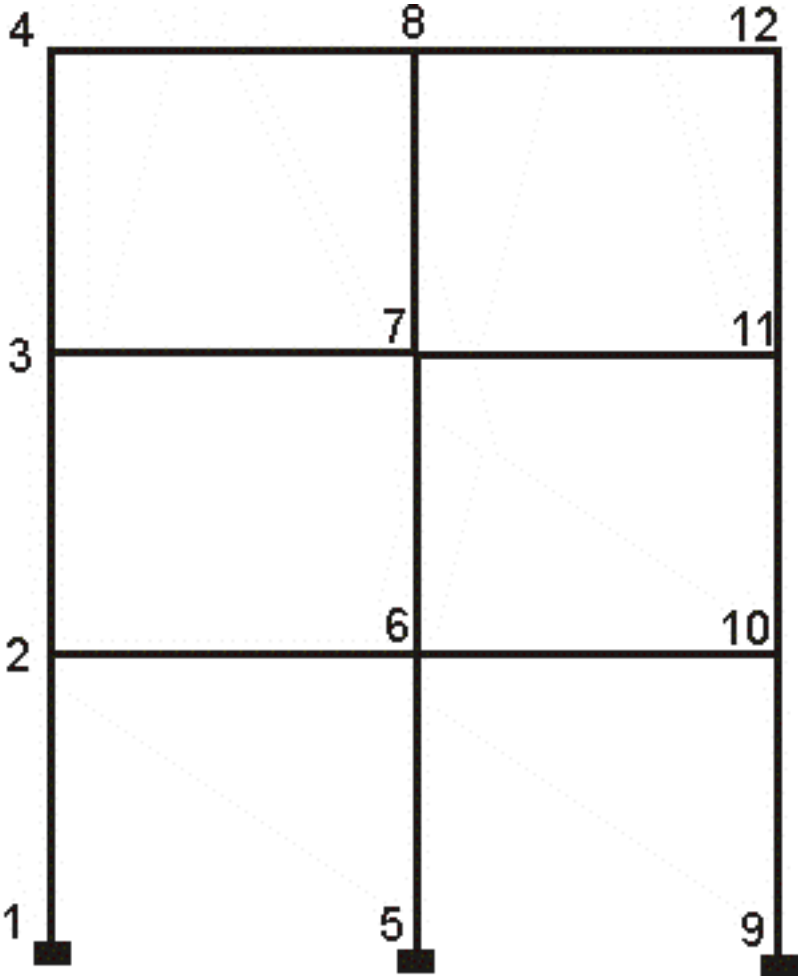
- **Liczne testy świadczą o to, że PARDISO wykazuje znacznie większą wydajność oraz speedup na komputerach wielordzeniowych w porównaniu do solwera wielofrontalnego.**

- **Formalnie PARDISO ma tryb OOC (out-of-core), jednak dla dużych zadań ten tryb nie działa.**

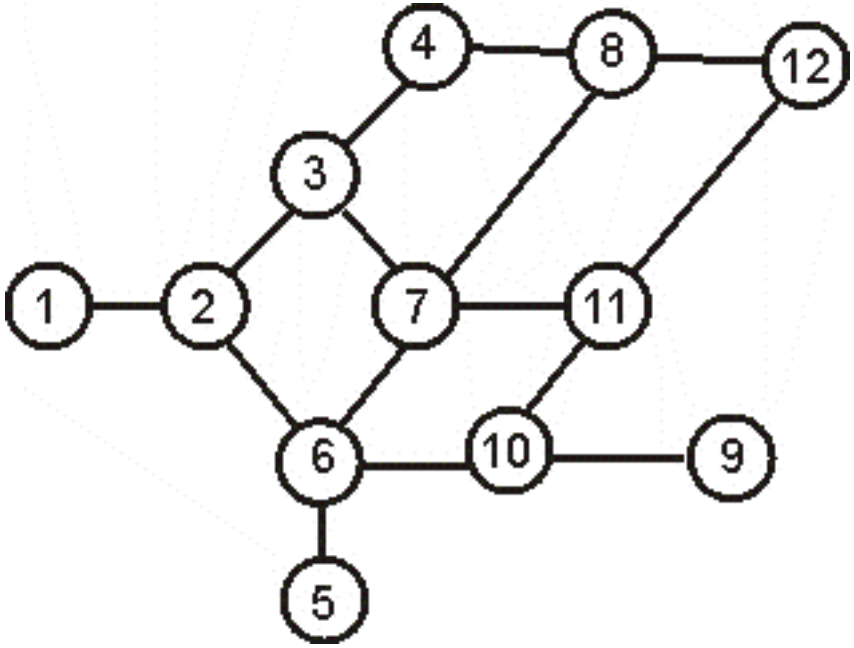
PARFES

- **Posłużyło to motywacją do opracowania solwera PARFES (Parallel Finite Element Solver), który**
  - **w trybie core mode (CM) powinien wykazywać wydajność i speedup, porównywalny do PARDISO**
  - **jeśli rozmiar zadania przekracza możliwości pamięci głównej, solwer powinien automatycznie przejść do trybu OOC – podłączyć dysk. Nawet w trybie OOC solwer powinien wykazywać stabilny speedup.**

Przykład ramy płaskiej

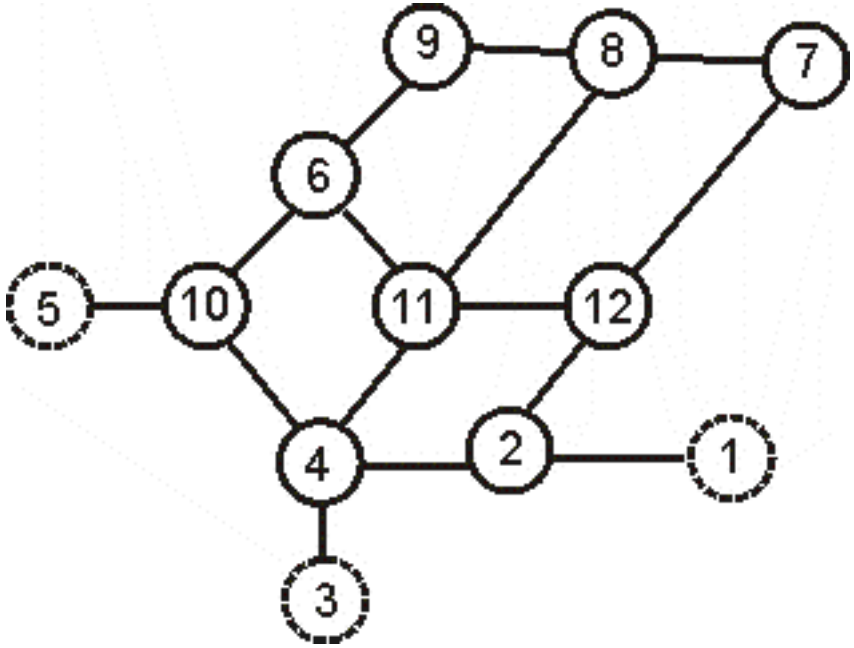


Rama płaska



Graf przyległości dla węzłów modelu obliczeniowego przed uporządkowaniem

Przykład ramy płaskiej

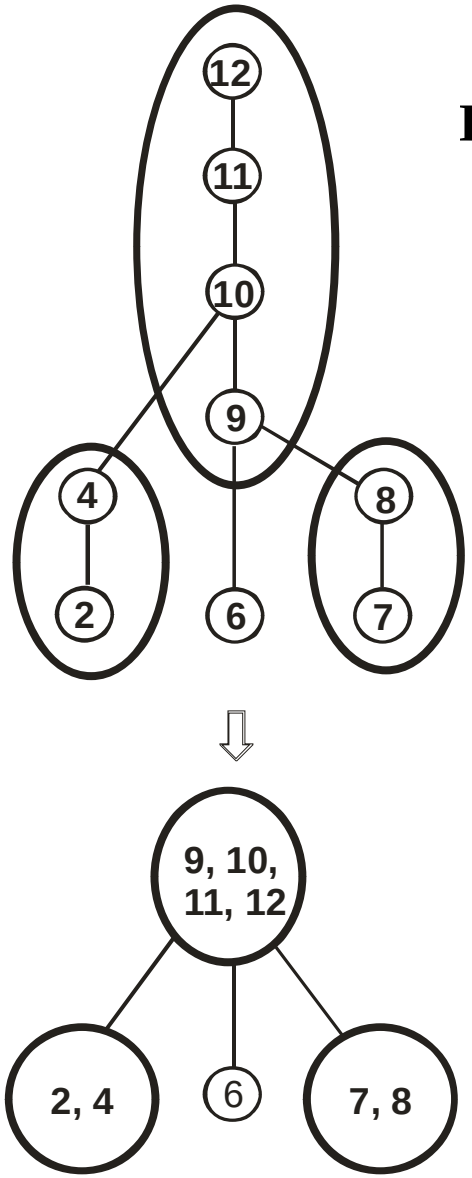


Graf przyległości po uporządkowaniu algorytmem MDA w wersji MMD. Podparte węzły są oznaczone linią przerywaną.

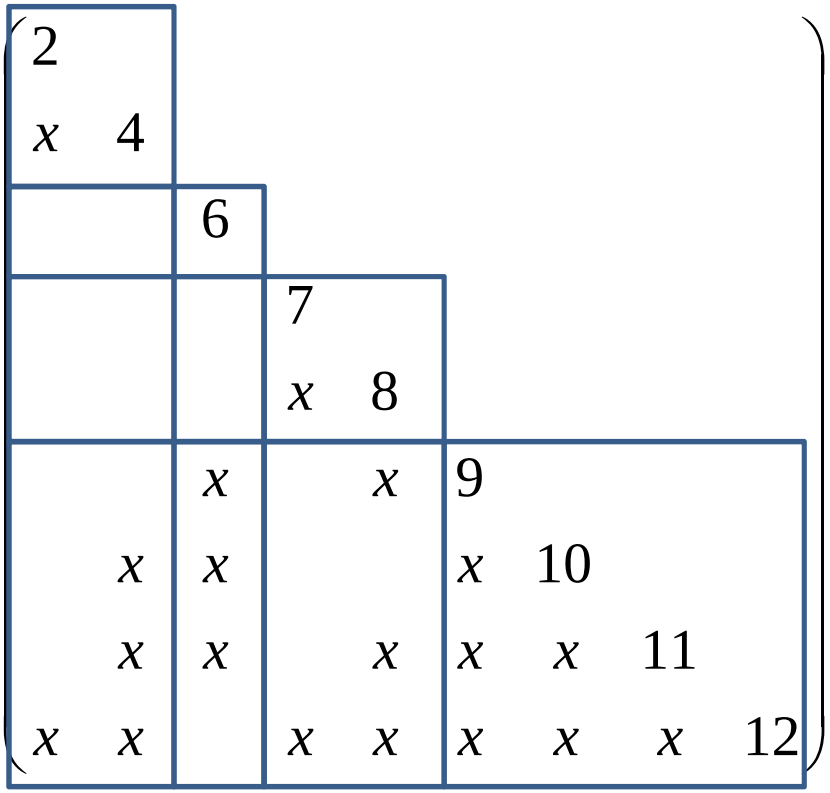
$$\begin{pmatrix}
2 & & & & & & & & \\
x & 4 & & & & & & & \\
 & & 6 & & & & & & \\
 & & & 7 & & & & & \\
 & & & x & 8 & & & & \\
 & & x & & x & 9 & & & \\
 & x & x & & & x & 10 & & \\
 & x & x & & x & x & x & 11 & \\
x & x & & x & x & x & x & x & 12
\end{pmatrix}$$

Niezerowa struktura macierzy rzadkiej po wykonaniu uporządkowania i faktoryzacji symbolicznej

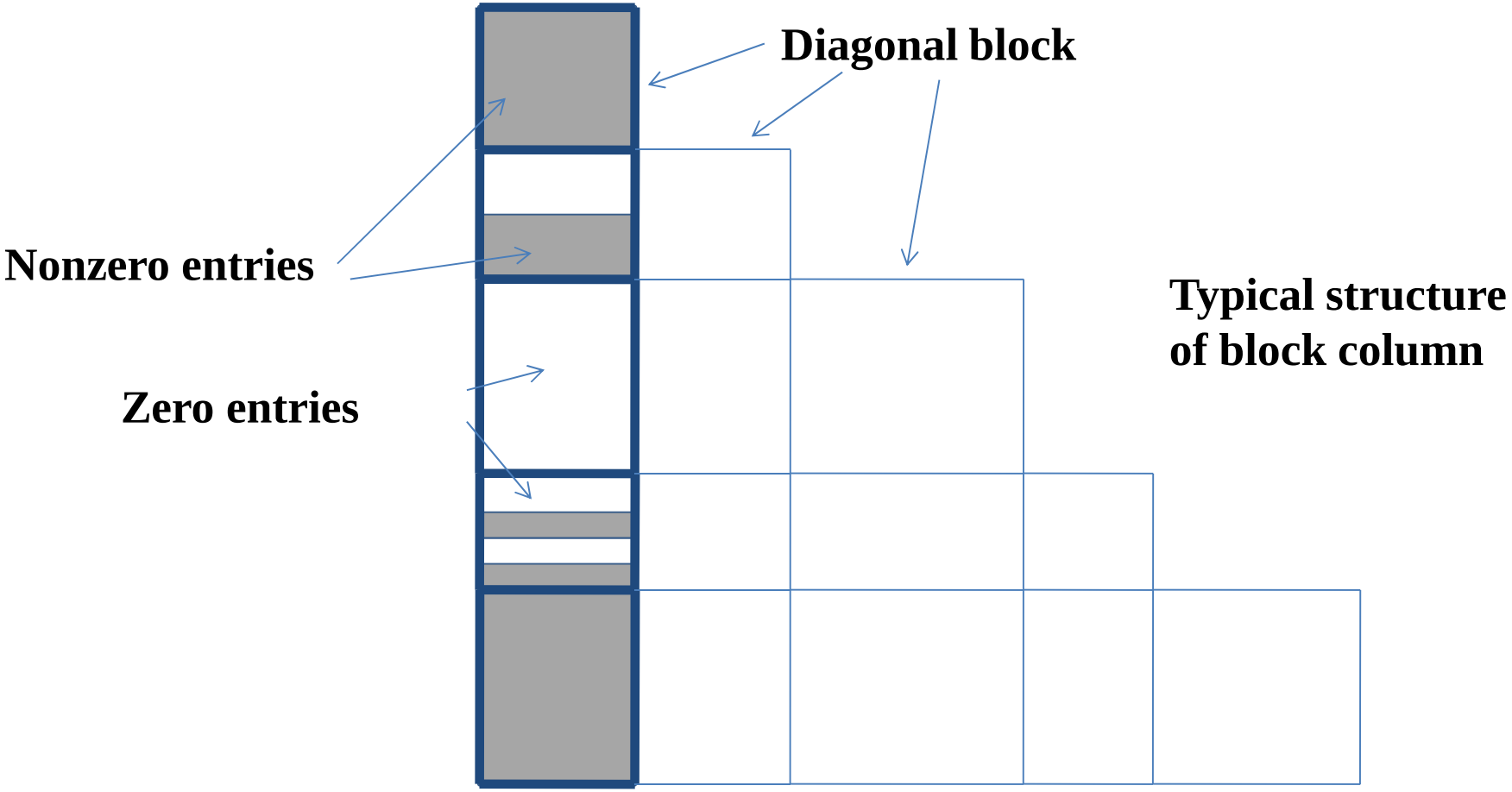
Trzeba podzielić macierz rzadką na prostokątne gęste bloki w taki sposób, żeby dołożyć minimalną ilość elementów zerowych – technika superwęzłowa



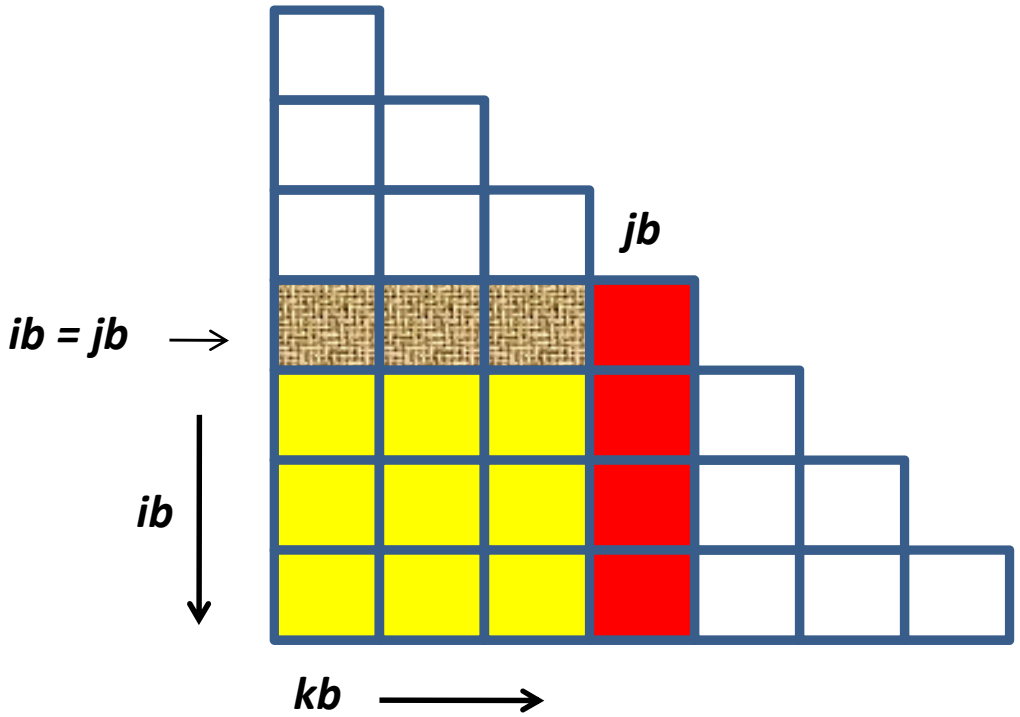
Drzewo eliminacji, drzewo superwęzłowe



Przygotowanie specjalnych struktur danych dla przechowywania prostokątnych gęstych podmacierzy



Faktoryzacja blokowa looking left



jb – kolumna blokowa, która jest sfaktoryzowana na podanym kroku. Przed faktoryzacją powinna być poprawiona przez kolumny, umieszczone zlewa.

Faktoryzacja blokowa looking left [3]

1. **if(core mode)**
   przygotuj kolumną blokową *jb* ϵ [1, N_b]
2. **do *jb* = 1, N_b**
3. **if(OOC ∨ OOC1)**
   przygotuj kolumną blokową *jb*
4. **Równoległe poprawienie kolumny *jb*:**

$$\mathbf{A}_{ib,jb} = \mathbf{A}_{ib,jb} - \sum_{kb \in List[\,jb\,]} \mathbf{A}_{ib,kb} \cdot \mathbf{S}_{kb} \cdot \mathbf{A}^{T}{}_{jb,kb}; \quad ib \geq jb, ib \in L$$

5. **Faktoryzacja kolumny blokowej *jb*:**

$$\mathbf{A}_{jb,jb} = \mathbf{L}_{jb,jb} \cdot \mathbf{S}_{jb} \cdot \mathbf{L}^{T}{}_{jb,jb}$$

$$\mathbf{L}_{jb,jb} \cdot \mathbf{S}_{jb} \cdot \mathbf{L}^{T}{}_{ib,jb} = \mathbf{A}^{T}{}_{ib,jb} \quad \rightarrow \quad \mathbf{L}_{ib,jb};$$

Faktoryzacja blokowa looking left

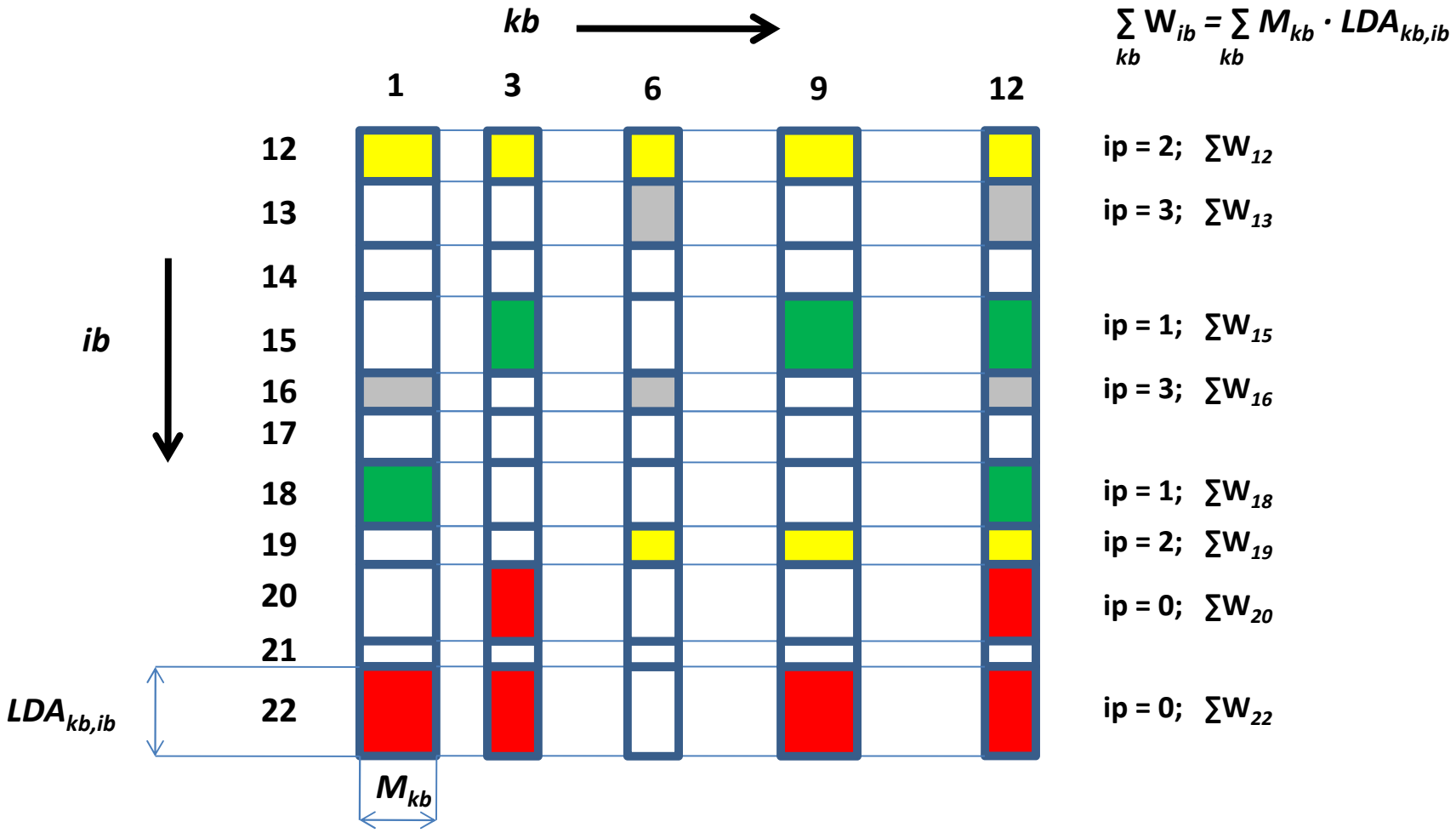
if(OOC ∨ OOC1)

zapisz kolumnę blokową *jb* na dysk i zwolni RAM dla wiersza blokowego *ib* = *jb*.

6. Dodaj *jb* do *List*[*lb*], *lb* > *jb*, jeśli kolumna *jb* poprawia kolumnę *lb*.

7. end do.

➢ Rzutowanie bloków $A_{ib,kb}$ na wątki [3]:



Sort: $W_{22} > W_{15} > W_{12} > W_{13} > W_{20} > W_{18} > W_{19} > W_{16}$

➢ Rzutowanie bloków $A_{ib,kb}$ na wątki [3]:

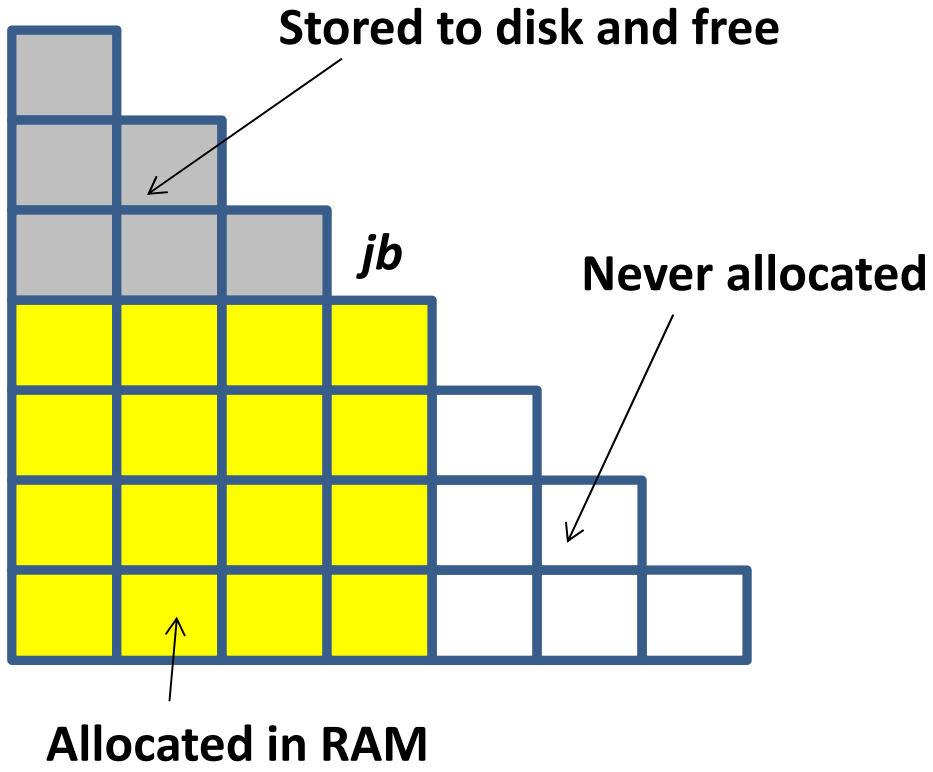
- **Defines the sum of weights:** $\sum_{kb} W_{ib} = \sum_{kb} M_{kb} \cdot LDA_{ib,kb}$
- **Descending sort of sum of weights**
- $\forall ip \in [0, 1, \ldots, \mathrm{Pr}ocNumb - 1] \quad sumofweights[ip] = 0$
- ***find min_ip*** **∈ [0, 1, …, ProcNumb-1]** **|** $\forall ib \in L_{jb}$ **(*sumofweights[min_ip]* is minimal)**
- ***sumofweights[min_ip] +=*** $\sum_{kb} W_{ib}$ ***; thread_numb[ib] = min_ip***
- ***end loop over ib***
- $\forall kb \in Link[jb]$ ***(loop over kb)***
- $\forall ib \in L_{kb}$ ***(loop over ib)***
- ***Q[thread_numb[ib]]*** ← **(**$A_{ib,kb}$ **;** $A_{jb,kb}$ ***; kb*****)** ***(put to queues Q[ip])***
- ***end loops over ib, kb***

➢ Równoległe poprawienie kolumny blokowej *jb*

- **# pragma omp parallel (*ip* ϵ *[0, ProcNumb-1]*)**
- **while(*Q[ip]* is not empty)**
- $\mathbf{A}_{ib,kb};\ \mathbf{A}_{jb,kb} \leftarrow Q[ip];\ Q[ip] \leftarrow (Q[ip] /(\mathbf{A}_{ib,kb};\ \mathbf{A}_{jb,kb}\ ;\ kb))$
- $\mathbf{A}_{ib,\,jb} = \mathbf{A}_{ib,\,jb} - \mathbf{A}_{ib,kb} \cdot \mathbf{S}_{kb} \cdot \mathbf{A}^{T}{}_{jb,kb};$
- **end while**
- **end of parallel region**

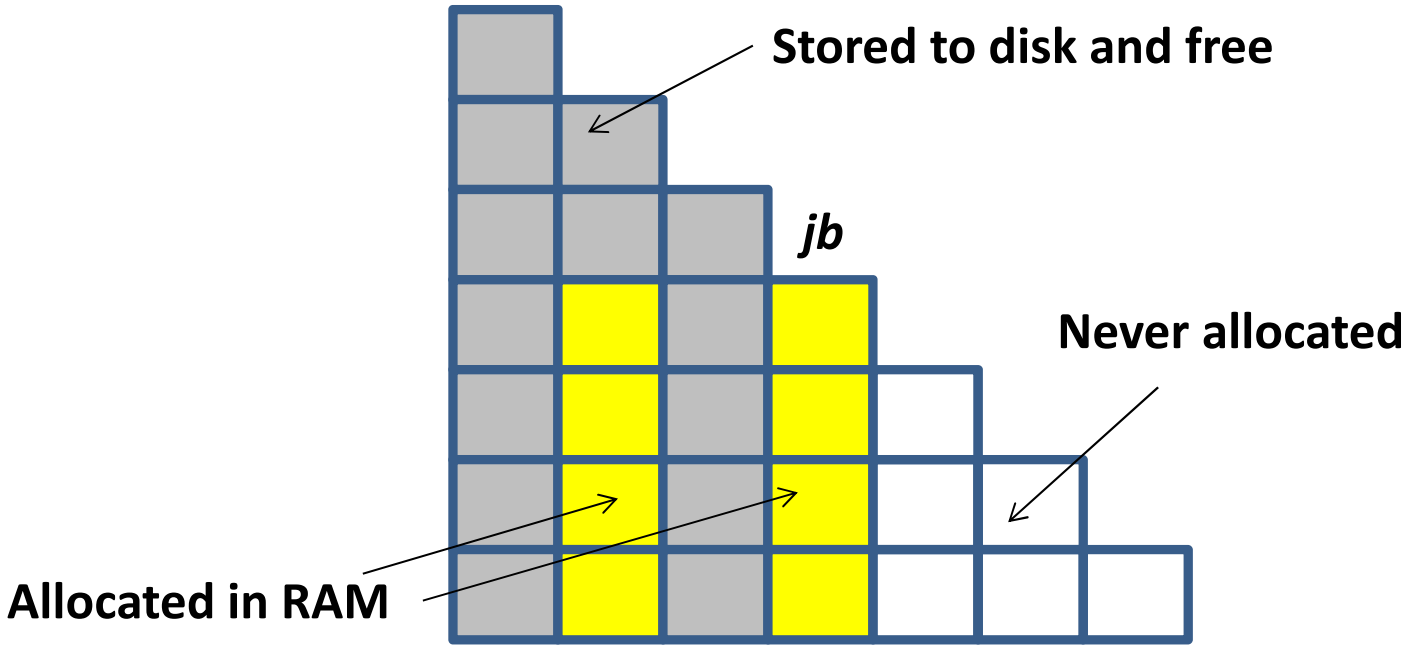
Wirtualizacja [3]

- **Tryb OOC włącza się automatycznie, jeśli rozmiar zadania przekracza możliwości RAM. Ten tryb doprowadzi do minimalnej ilości odwołań do dysku.**



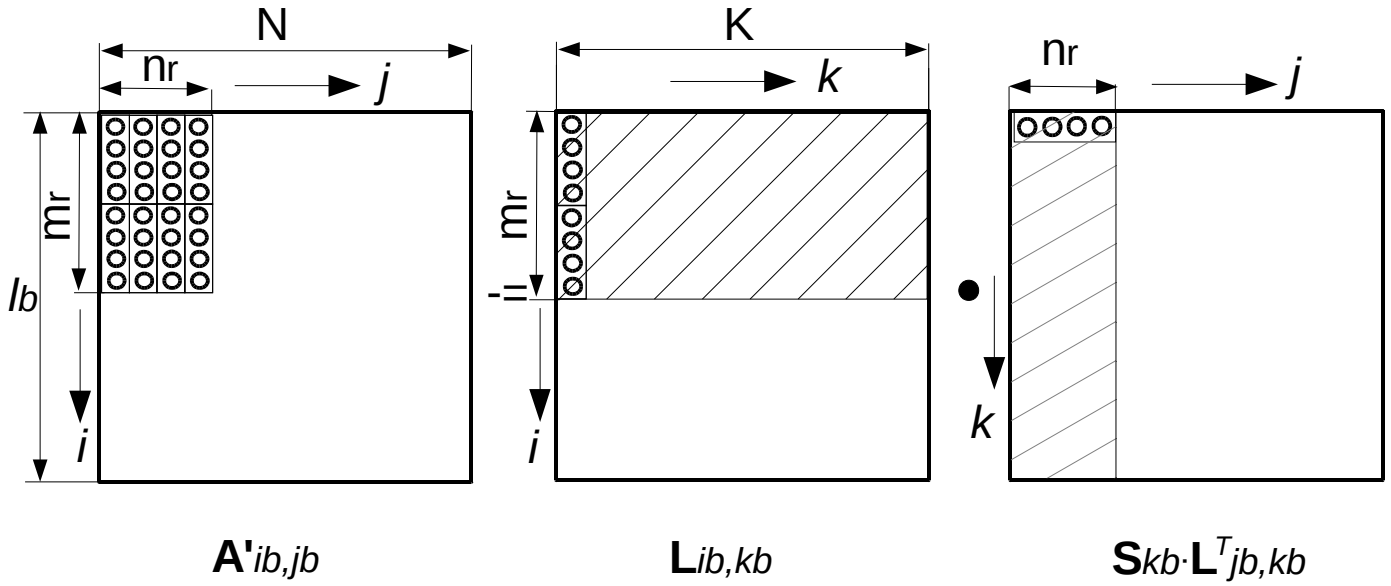
Wirtualizacja [6]

- **Tryb OOC1 włącza się automatycznie, jeśli rozmiar zadania przekracza możliwości trybu OOC. Ilość odwołań do dysku istotnie się zwiększa, jednak powstaje możliwość rozwiązywać duże zadania na komputerach z małą pamięcią RAM.**



Wektoryzowanie (AVX) [7]

- **Opracowane specjalnie dla PARFES mikrojądro microkernel_8x4_AVX na podstawie techniki AVX na procesorach architektury AMD Opteron istotnie przyspiesza wydajność PARFES w porównaniu do zastosowania dgemm z Intel MKL oraz dgemm z ACML (AMD Core Math Lib).**

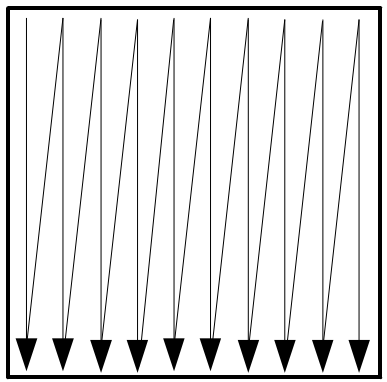


$\mathbf{A'}_{ib,jb}$ $\mathbf{L}_{ib,kb}$ $\mathbf{S}_{kb} \cdot \mathbf{L}^{T}_{jb,kb}$

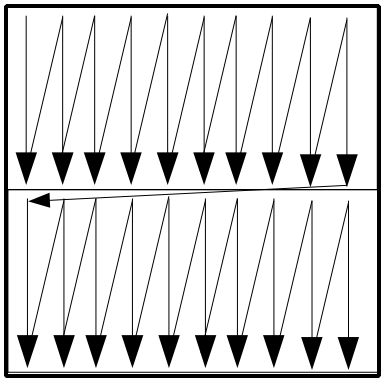
Blokowanie rejestrów YMM w blokach lb ≤ 120

Wektoryzowanie (AVX) [7]

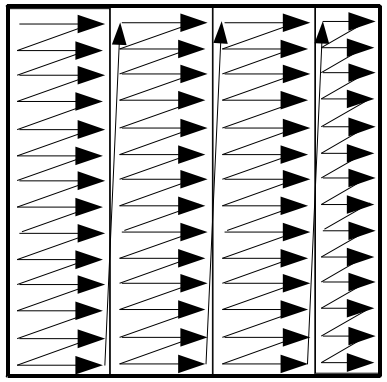
Przepakowywanie danych:



$\mathbf{A'}_{ib,jb}$



$\mathbf{L}_{ib,kb}$

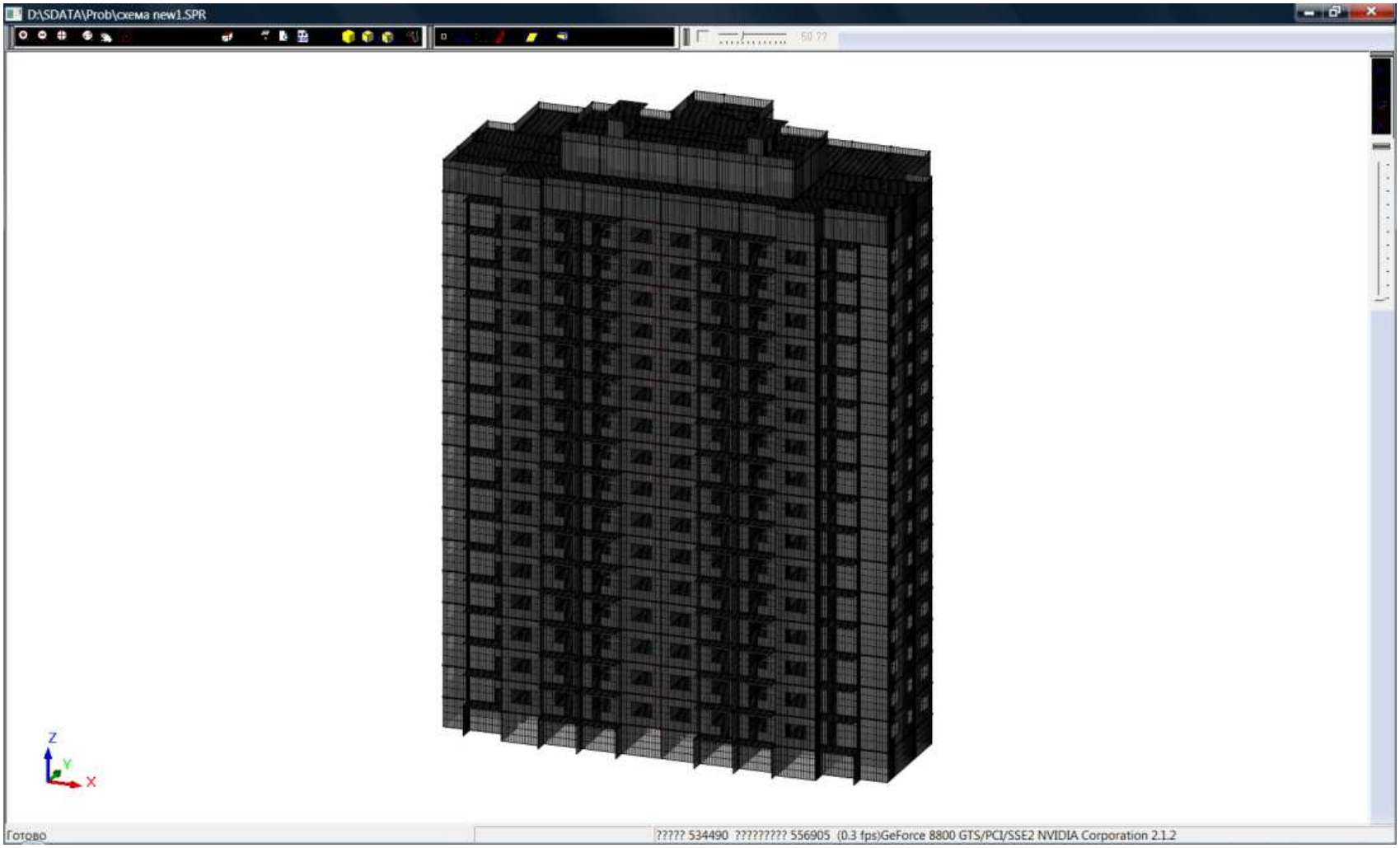


$\mathbf{S}_{kb} \cdot \mathbf{L}^{T}_{jb,kb}$

Microkern_8×4_AVX: matrix block multiplication

```
for(j=0; j<N; j+=n_r)
    for(i=0; i<l_b; i+=m_r)
        A´_ij^ib,jb += L_i^ib,kb • B_j^jb,kb
```

Wyniki numeryczne



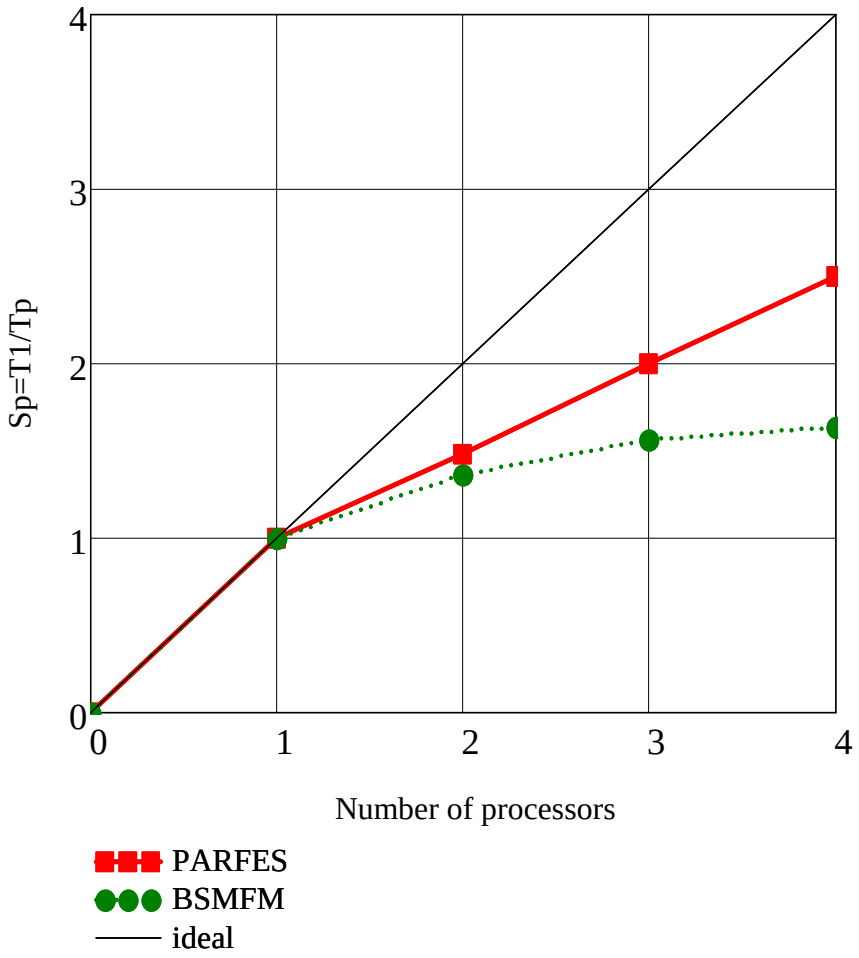
Zadanie schema_new_1, 3 198 609 równań

Wyniki numeryczne

Tabela 3. Trwałość etapów rozwiązania zadania schema_new_1 problem (3,198,609 equations) na komputerze z procesorem a Core™2 Quad [3]

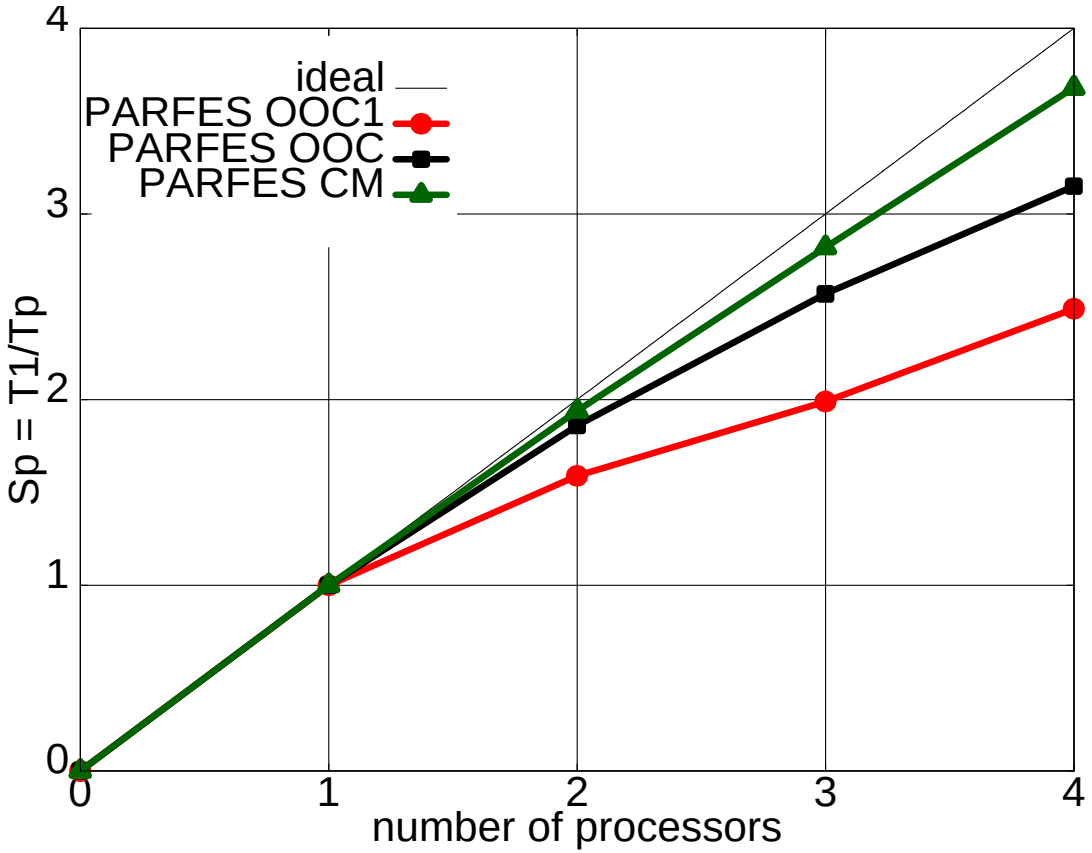
Method	NonZer (L), MB	Ana-lysis, s	Numerical factorization, s				Solution phase, s		Com-ments
			Number of processors				Number of proc.		
			1	2	3	4	1	4	
PARFES (OOC)	12 186	23.6	1 190	802	594	475	804	526	X64
PARDISO(OOC)	10 662	61.4	Numer. factorization phase: error = -11						X64
BSMFM (OOC)	10 869	9.0	2 011	1 482	1 286	1 232	497		x64

Wyniki numeryczne: tryb OOC [3]



Zadanie Schema_new_1 na komputerze z procesorem Core™2 (numerical factorization phase)

Wyniki numeryczne



32 453 MFLOPS
12 186 MB RAM

22 898 MFLOPS
2 995 MB RAM

8 610 MFLOPS
706 MB RAM

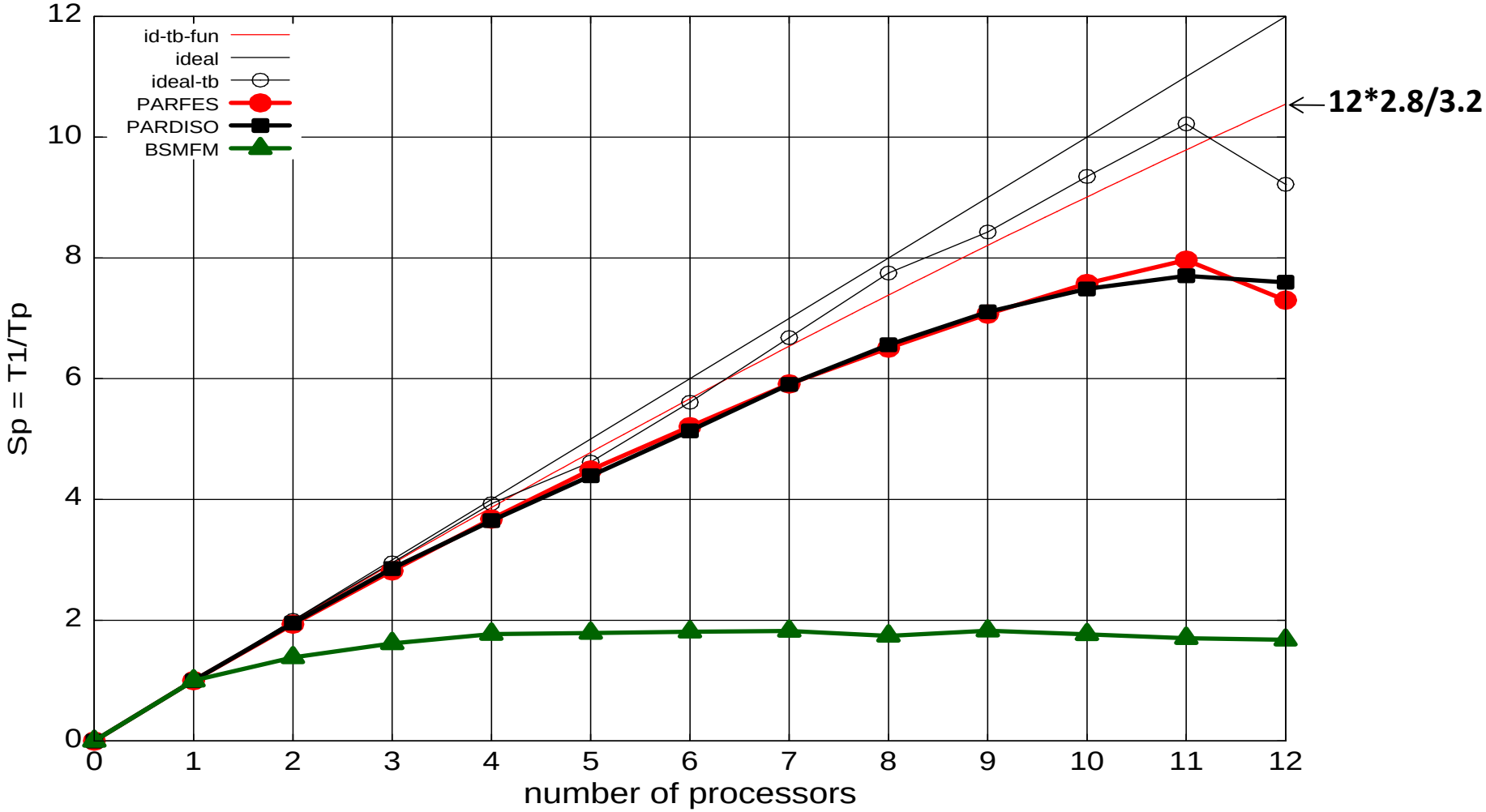
Zadanie Schema_new_1 problem na komputerze z procesorem AMD Phenom™ II x4 995 (numerical factorization phase), RAM - 16 GB

Wyniki numeryczne

Tab. 8. Trwałość etapu faktoryzacji zadania schema_new_1 (3,198,609 równań), stacja robocza DELL z dwoma procesorami Intel Xeon X5660 @ 2.8 GHz /3.2 GHz (12 cores), RAM 24 GB, DDR3, Core mode, platform ×64

No's of proc.	PARFES		PARDISO		BSMFM	
	Anal., s	Num. Fact., s	Anal., s	Num. Fact., s	Anal., s	Num. Fact., s
1	16.9	654	31.06	596	13	1406
2	16.9	337.8	23.59	305.3	13	1015
3	16.9	232.1	25.33	208.6	13	869
4	16.9	177.9	23.26	163.3	13	793
5	16.9	145.7	23.79	135.7	13	786
6	16.9	125.6	25.68	116	13	777
7	16.9	110.6	23.11	100.9	13	772
8	16.9	100.5	23.43	90.83	13	807
9	16.9	92.5	23.98	83.85	13	770
10	16.9	86.3	23.71	79.6	13	796
11	16.9	82.1	29.86	77.36	13	825
12	16.9	87.5	28.58	78.45	13	839

Wyniki numeryczne



Zadanie Schema_new_1 na komputerze z procesorami Intel Xeon X5660 @ 2.8 GHz /3.2 GHz (numerical factorization phase) - CM

Wyniki numeryczne

Notebook Toshiba Satellite:

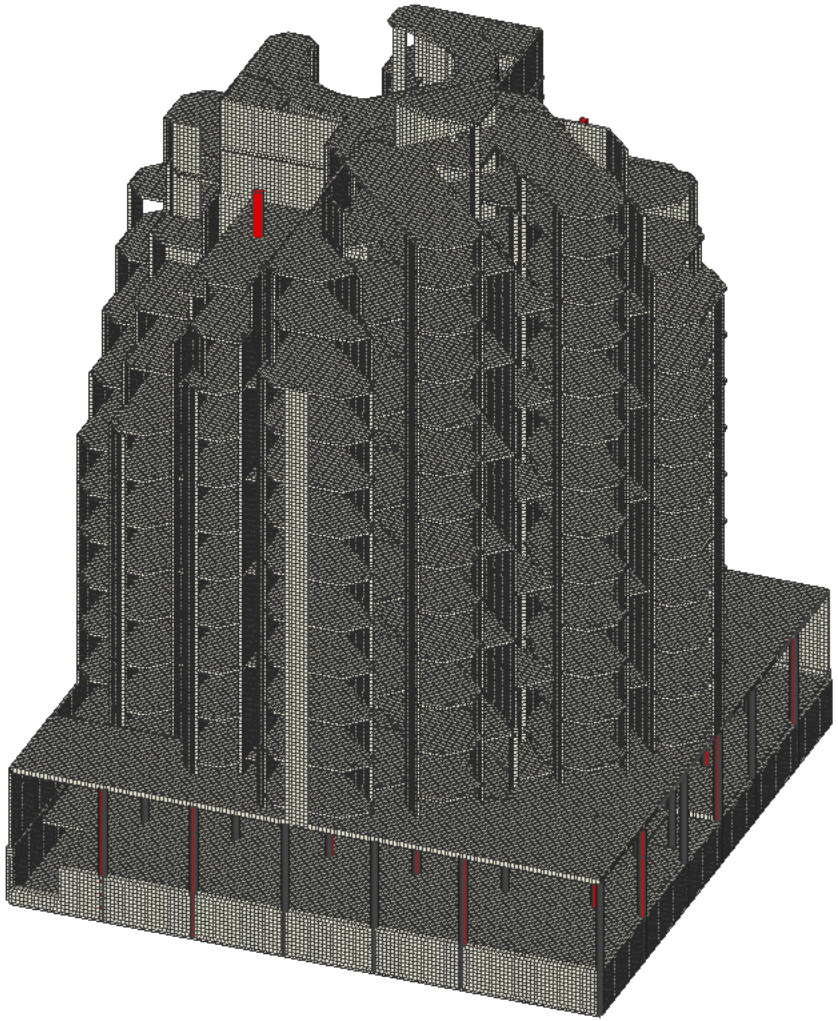
Processor: Intel Pentium Dual CPU T3200 @ 2.00 GHz
Cache: L1: 32 KB, L2: 1024 KB
RAM: DDR2 – 667 MT/s 4 GB

Chipset: Intel GL40 rev. 07
OS – Windows Vista™ Business (64-bit), Service Pack 2

Application ia32 OOC1 mode 2 threads

Analysis	**:**	**26 s**
Assembling	**:**	**196 s = 3 ‘ 16”**
Numerical factoring	**:**	**3 111 s = 51’ 51”**
Forward/Back reduction	**:**	**1 194 s = 19‘ 54”**
Total time	**:**	**4 532 s = 75’ 32”**

Wyniki numeryczne

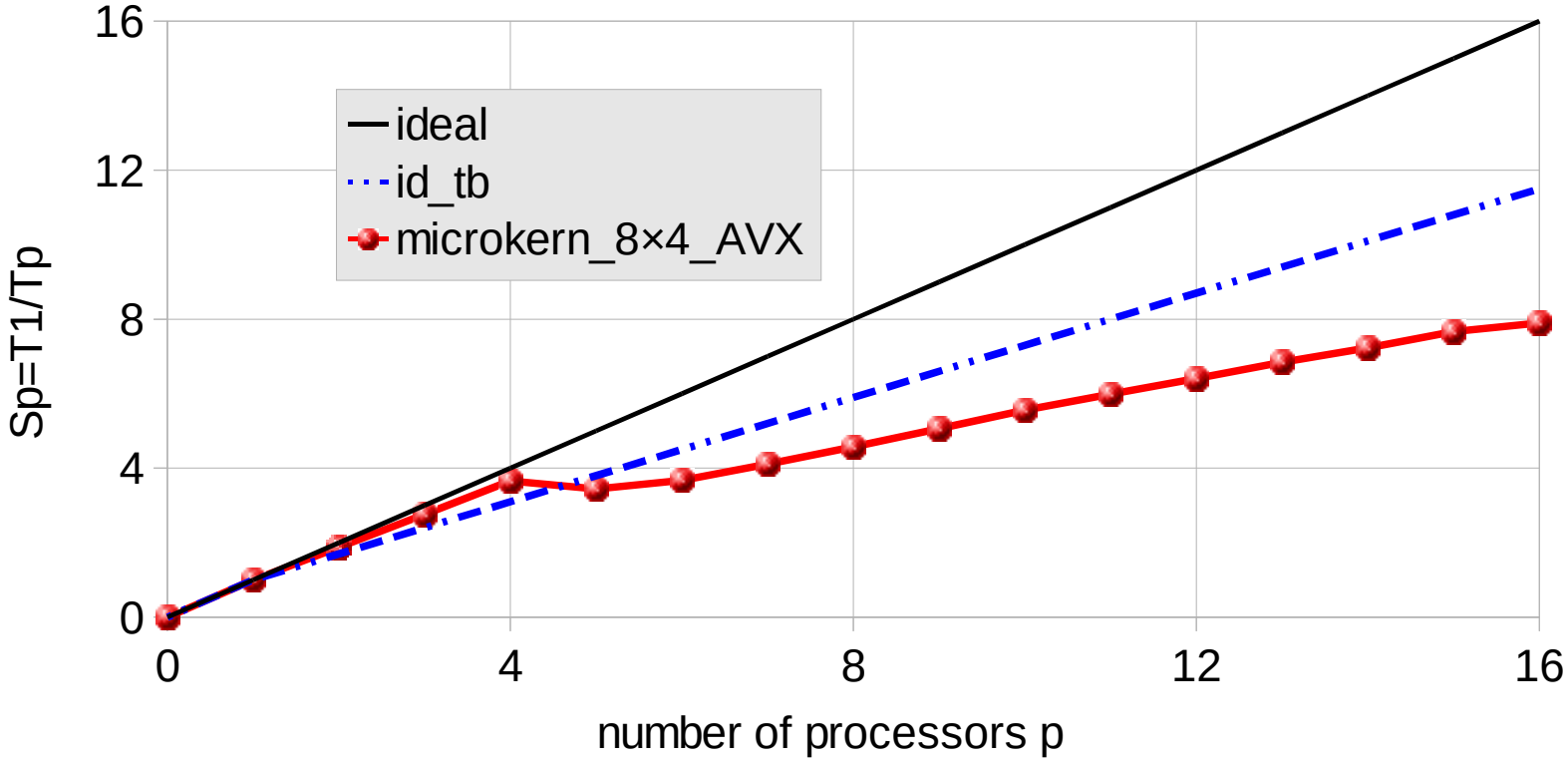


Duży model obliczeniowy budynku biurowego

Ilość równań: 7 328 394

Komputer:
16-core processor AMD Opteron 6276, 2.3/3.2 GHz, 64 GB DDR3 RAM, OS Windows Server 2008 R2 Enterprise SP1, 64 bit.

Wyniki numeryczne



ideal – przyspieszenie idealne

id_tb – oszacowanie idealnego przyspieszenia z uwzględnieniem TurboBust

microkern_8x4_AVX – przyspieszenie rzeczywiste

Wyniki numeryczne

Method	Total solution time, seconds	Number of threads	RAM mode
PARFES	1 294/2 568	16	CM
BSMFM	9 935	16	CM
PARDISO	page fault in analysis phase		CM
PSICCG	2 006	5	CM

PARFES:

1 294 s – czas rozwiązania zadania przy użyciu mikrojądra microkern_8x4_AVX

2 568 s - czas rozwiązania zadania przy użyciu dgemm z biblioteki Intel MKL 10.2.2.025

REFERENCES

1. **Amestoy PR, Duff IS, L'Excellent J-Y. Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Meth. Appl. Mech. Eng., 184: 501–520, 2000.**
2. **Dobrian F, Pothen A. Oblio: a sparse direct solver library for serial and parallel computations. Technical Report describing the OBLIO software library, 2000.**
3. **Fialko S. PARFES: A method for solving finite element linear equations on multi-core computers. Advances in Engineering software. v 40, 12, 2010, pp. 1256 – 1265.**
4. **Fialko S. The block substructure multifrontal method for solution of large finite element equation sets. Technical Transactions, 1-NP, issue 8: 175 – 188, 2009.**
5. **Fialko S. The direct methods for solution of the linear equation sets in modern FEM software. Moscow: SCAD SOFT , 2009. (in Russian).**

REFERENCES

6. Fialko S. Parallel Finite Element Solver for Multi-Core Computers. Federated Conference on Computer Science and Information Systems, September 9–12, 2012. Wrocław, Poland. IEEE Xplore Digital Library, 978-83-60810-51-4, IEEE Catalog Number CFP1285N-USB. P. 525 – 532. URL: http://fedcsis.org/proceedings/fedcsis2012/pliks/101.pdf

7. Fialko S. Application of AVX (Advanced Vector Extensions) for Improved Performance of the PARFES – Finite Element Parallel Direct Solver. Federated Conference on Computer Science and Information Systems, September 8–11, 2013. Kraków, Poland. IEEE Xplore Digital Library, pp. 447 – 454. https://fedcsis.org/proceedings/2013/pliks/98.pdf

8. Fialko S. Parallel direct solver for solving systems of linear equations resulting from finite element method on multi-core desktops and workstations. Computers and Mathematics with Applications 70 (2015) 2968–2987. doi:10.1016/j.camwa.2015.10.009

REFERENCES

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