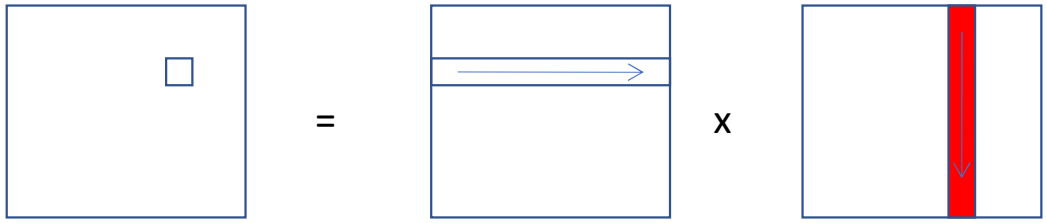
Lokalność dostępu do danych

Na przykładzie mnożenia tablic

Opracował Rafał Walkowiak

```
for ( int i = 0 ; i < n ; i++)
        for (int j = 0 ; j < n ; j++)
                for (int k = 0 ; k < n ; k++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```

IJK



Tablica C – dla każdego słowa w wierszu dostęp podwójny (odczyt i zapis)

Tablica A – każdy wiersz N razy czytany

Tablica B – tablica N razy czytana kolumnami

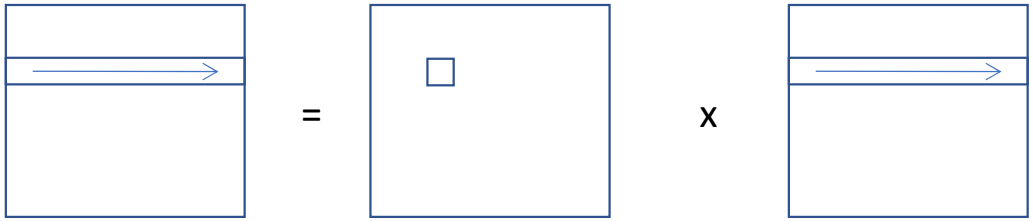
Podstawowy problem to brak lokalności przestrzennej (tablica B) – możliwe że każda kolejna instrukcja to odwołanie do innej strony wirtualnej, a zbiór używanych stron wirtualnych będzie typowo zbyt duży, aby ich adresy były dostępne w DTLB – wynik to n^3 nieefektywnych dostępów (braki trafień do DTLB).

Brak lokalności czasowej tablica B gdy wielokrotnie czytana tablica B > ppp

ppp - wielkość pamięci podręcznej procesora

```
for ( int i = 0 ; i < n ; i++)
        for (int k = 0 ; k < n ; k++)
                for (int j = 0 ; j < n ; j++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```

IKJ



Tablica C – dla każdego słowa w wierszu N razy dostęp podwójny (odczyt i zapis)

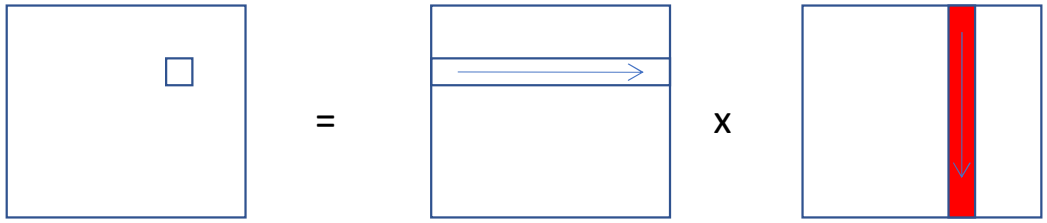
Tablica A – każdy wiersz 1 raz czytany

Tablica B – tablica N razy czytana wierszami

Lokalność przestrzenna nie jest problemem (czytanie tablic wierszami), problem stanowi sytuacja gdy B (n razy odczytywana w kodzie tablica) jest większa od dostępnej pamięci podręcznej procesora – wtedy brak lokalności czasowej. Adresy stron dla tablicy B również mogą nie mieścić się w pamięci, ale odwołania wierszami do B nie powodują trudności gdyż wtedy liczba braków trafień do DTLB nie jest równa n*n*n (liczba dostępów do tablicy) lecz jest tylko iloczynem n (liczby odczytów tablicy B) i liczby stron wirtualnych z tablicą B.

```
for (int j = 0 ; j < n ; j++)
        for ( int i = 0 ; i < n ; i++)
                for (int k = 0 ; k < n ; k++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```

JIK



Tablica C – dla każdego słowa w kolumnie dostęp podwójny (odczyt i zapis)

Tablica A – tablica N razy czytana wierszami

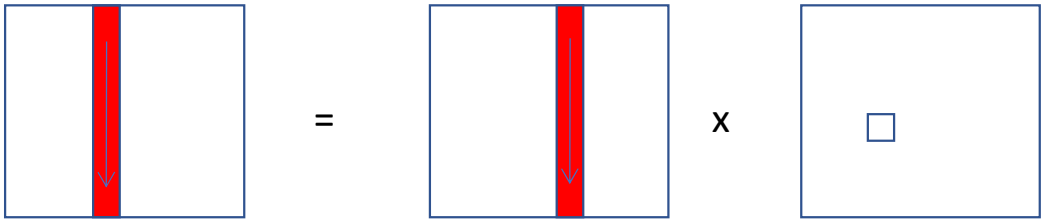
Tablica B – każda kolumna N razy czytana

Problemem jest brak lokalności przestrzennej (tablice B i C) – możliwe, że każda kolejna instrukcja to odwołanie do innej strony wirtualnej (tablica B) , a zbiór stron wirtualnych będzie zbyt duży aby ich adresy były dostępne w DTLB. $n^3 + n^2$ – nieefektywnych dostępów (braki trafień do DTLB)

W tym przypadku brak lokalności czasowej dostępu do tablicy A nie stanowi już wielkiego problemu gdyż ta nieefektywność ginie w powyżej opisanej!

```
for (int j = 0 ; j < n ; j++)
        for (int k = 0 ; k < n ; k++)
                for ( int i = 0 ; i < n ; i++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```

JKI



Słowa z k- tej kolumny A mnożone przez słowo B z k-tego wiersza i j-tej kolumny

Tablica C – dla każdego słowa w kolumnie N razy dostęp podwójny (odczytu i zapis)

Tablica A – tablica N razy czytana kolumnami

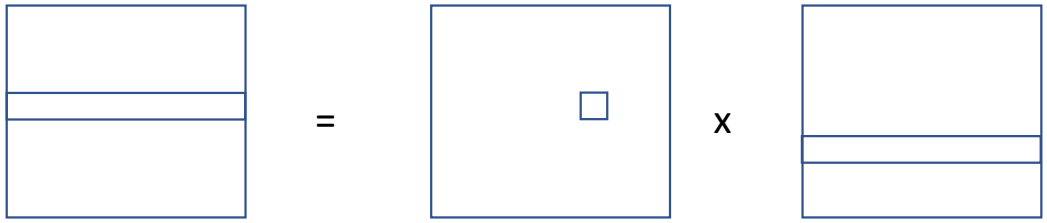
Tablica B – tablica 1 raz czytana kolumnami

Problemem jest brak lokalności przestrzennej (wszystkie tablice)– możliwe, że każda kolejna instrukcja to odwołanie do 2 nowych stron wirtualnych (tablica A,C), a zbiór stron wirtualnych będzie zbyt duży aby ich adresy były dostępne w DTLB. $2 * n^3 + n^2$ nieefektywnych dostępów (braki trafień do DTLB dla C, A,B).

W tym przypadku brak lokalności czasowej dostępu do tablicy A nie stanowi już wielkiego problemu gdyż ta nieefektywność ginie w powyżej opisanej!

```
for (int k = 0 ; k < n ; k++)
        for ( int i = 0 ; i < n ; i++)
                for (int j = 0 ; j < n ; j++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```

KIJ



Słowo z k- tej kolumny A mnożone przez słowa z k-tego wiersza B

Tablica C – tablica dostępy podwójne N razy wierszami

Tablica A – każda kolumna 1 raz czytana

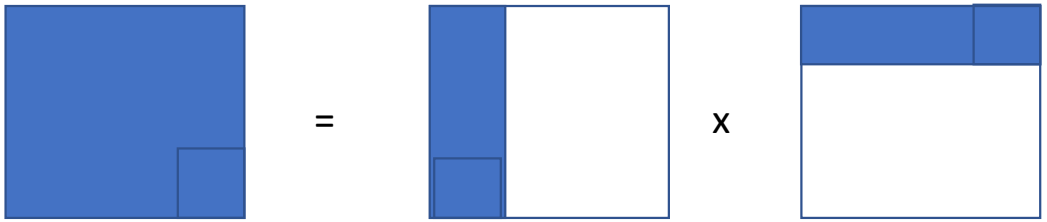
Tablica B – każdy wiersz N razy czytany

Problemem jest brak lokalności czasowej (tablica C).

Nie nadaje się do obliczeń równoległych jeżeli podział pracy na pętli zewnętrznej ze względu na wyścig powodowany jednoczesnym dostępem do tablicy C. Podział pracy na pętlach wewnętrznych wprowadza cykliczne bariery synchronizacyjne, które są konieczne.

KIJ

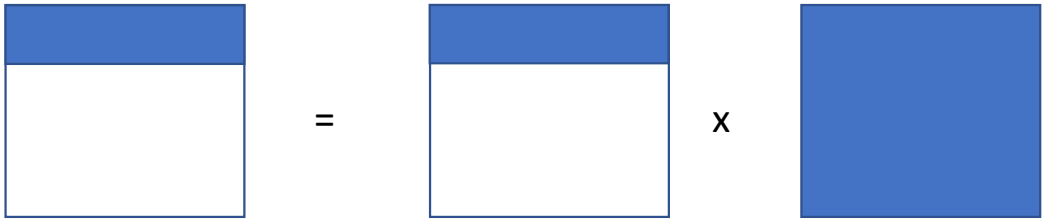
```
#pragma omp parallel for
for (int k = 0 ; k < n ; k++)
        for ( int i = 0 ; i < n ; i++)
                for (int j = 0 ; j < n ; j++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```



Kod jest scharakteryzowany wyścigiem w dostępie do danych.
Na niebiesko zaznaczono dane używane przez jeden wątek.
Zaznaczono obliczane i mnożone części tablic przez jeden wątek.

KIJ

```
for (int k = 0 ; k < n ; k++)
#pragma omp parallel for //deterministyczne szeregowanie statyczne
        for ( int i = 0 ; i < n ; i++)
                for (int j = 0 ; j < n ; j++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```

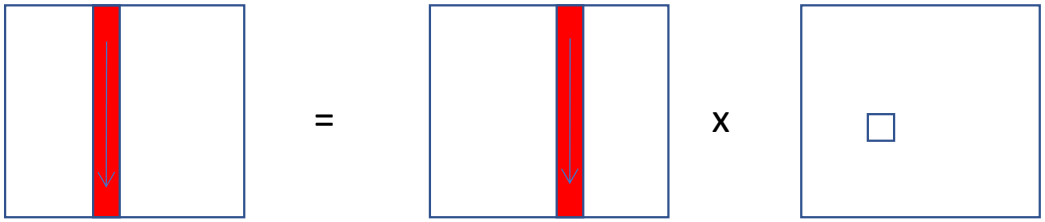


Kod zawiera n synchronizacji.

Na niebiesko zaznaczono dane używane przez jeden wątek.

```
for (int k = 0 ; k < n ; k++)
        for (int j = 0 ; j < n ; j++)
                for ( int i = 0 ; i < n ; i++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```

KJI



Tablica C – tablica dostępy podwójne N razy kolumnami

Tablica A – każda kolumna N razy czytana

Tablica B – tablica raz czytana wierszami

Problemem jest brak lokalności przestrzennej (tablice C i A) – możliwe, że każda kolejna instrukcja to odwołanie do 2 nowych stron wirtualnych, a zbiór stron wirtualnych będzie zbyt duży aby ich adresy były dostępne w DTLB $2 * n^3$ – nieefektywnych dostępów.

Nie nadaje się do obliczeń równoległych, gdyż podział pracy na pętli zewnętrznej wprowadza do kodu wyścig w dostępie do danych.

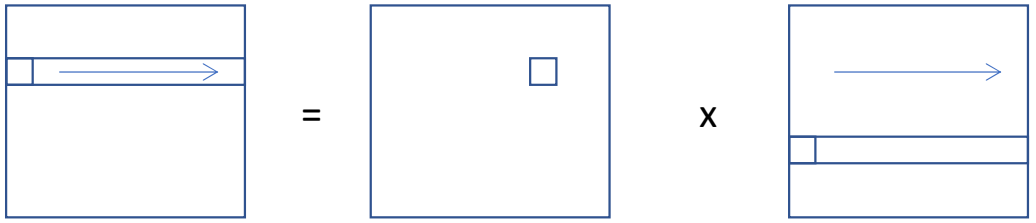
Kolejność pętli	Dostępy do C[i][j]	Dostępy do A[i][k]	Dostępy do B[k][j]
ijk	1x wierszami	wiersz nx	Nx kolumnami
ikj	Wiersz nx	1x wierszami	Nx wierszami
jik	1x kolumnami	Nx wierszami	Kolumna nx
jki	Kolumna nx	Nx kolumnami	1x kolumnami
kij	Nx wierszami	1x kolumnami	Wiersz nx
kji	Nx kolumnami	Kolumna nx	1x wierszami

Z zestawienia zbiorczego można wyciągnąć następujące wnioski:

- W każdej metodzie jedna z tablic jest wielokrotnie w całości wykorzystywana powodując potencjalnie brak lokalności czasowej dostępu
- Dostęp kolumnami szczególnie wielokrotny do tych samych danych może powodować kosztowne przetwarzanie (wewnętrzna pętla i)
- Korzystana jest kolejność ikj ze względu na sposób dostępu do danych – lokalność przestrzenna dostępu – minimalizacja liczby braków trafień do DTLB.

Mnożenie macierzy kolejność pętli IKJ - dlaczego tak szybko ?

```
float a[N,N], b[N,N], c[N,N]
 for i=0; i<N; i++
     for k=0; k<N; k++
          for j=0;j<N; j++
                         c[i,j]+= a[i,k]*b[k,j];
```



Wykonane jako:

a) mnożenie słowa tablicy **a** przez wszystkie elementy wiersza **b** i dodanie wyników do wiersza **c**

b) mnożenie kolejnego słowa z wiersza tablicy **a** przez kolejny wiersz **b** i dodanie wyników do elementów wiersza **c**

c) obliczanie w sposób a) b) pozostałych wierszy

Cecha podejścia to (bez wektoryzacji):

- jednokrotny odczyt każdego elementu tablicy **a** – N*N odczytów
- odczyty sąsiednich słów tablicy **b** – każde słowo użyte dla obliczenia wiersza wyniku - razem N*N*N odczytów słów tablicy **b**
- odczyty i zapisy sąsiednich słów tablicy **c** – w każdej iteracji – razem N*N*N odczytów i zapisów

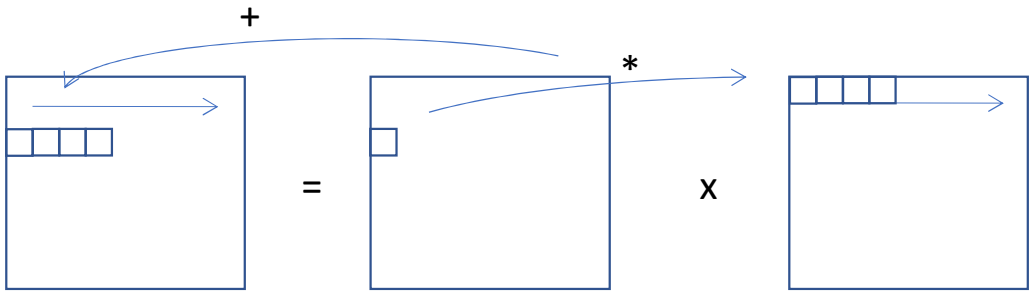
Zastosowanie instrukcji SSE

- SSE - SIMD streaming extension – jednakowa operacja wykonywana na wielu obiektach danych
- rejestry 128 bitowe dla zapisu/obliczeń na 4 słowach 4 bajtowych (pojedynczej precyzji) 4*4*8
- Mnożenie macierzy -zamiast mnożyć pojedyncze słowa mnożenie słów z wektorów 4 elementowych – wynik wektor
- Użyte przez kompilator instrukcje (obliczenia pojedynczej precyzji) :
  - movaps (adres1), xmm0 - odczyt z pamięci 4 sąsiednich słów pojedynczej precyzji - 128 bitów – 4x mniej dostępów do tablicy B
  - shufps 0,xmm1,xmm1 – powielenie jednego słowa 32 bit w 4 słowa w tym samym rejestrze
  - mulps xmm1, xmm0 – mnożenie 4 par słów – 4 x mniej instrukcji mnożenia
  - addps (adres2), xmm0 – sumowanie 4 par słów – 4x mniej instrukcji dodawania i 4 x mniej odczytów z tablicy C
  - movaps xmm0, (adres2) – zapis do pamięci 4 sąsiednich słów – 128 bitów - 4x mniej zapisów do tablicy C

Mnożenie macierzy kolejność pętli IKJ

```
float a[N,N], b[N,N], c[N,N]

for i=0; i<N; i++
    for k=0; k<N; k++
        for j=0;j<N; j=j+4
                    c[i,j:j+3]+= a[i,k]*b[k,j:j+3];
```



Wykonane jako:

a) za pomocą jednego rozkazu wektorowego mnożenie słowa tablicy **a** przez 4 kolejne słowa z wiersza tablicy **b** i dodanie 4 wyników do kolejnych słów wiersza **c**

b) za pomocą wielu rozkazów wektorowych mnożenie słowa tablicy **a** przez następne 4 kolejne słowa z wiersza tablicy **b** i dodanie 4 wyników do odpowiednich słów wiersza **c**

c) mnożenie kolejnego słowa z wiersza tablicy **a** przez kolejne wiersz **b** i dodanie wyników do elementów wiersza **c**

d) obliczanie w sposób a) b) pozostałych wierszy

Cecha podejścia to

- jednokrotny odczyt każdego elementu tablicy **a** – N*N odczytów
- odczyty sąsiednich 4 słów tablicy **b** - **wektor** – każde słowo użyte dla obliczenia wiersza wyniku - razem N*N*N/4 odczytów wektorów słów tablicy **b**
- odczyty i zapisy wektorów słów tablicy **c** – w każdej iteracji – razem N*N*N/4 odczytów i zapisów

Kod asemblera dla zakresu pętli wewnętrznej programu

```
; for j=0;j<N;j ++ c[i,j]+= a[i,k]*b[k,j];  ;wartość  a[i,k] jest stała
                                            ;%rcx, %rax to adres początku wiersza tablicy B
e2: movaps (%rcx,%rax), %xmm0               ; odczyt 4 słów tab B, wyrównany do granicy 16 bajtów
mulps %xmm1, %xmm0                          ; iloczyn 4 par słów
addps   (%rdx, %rax), %xmm0                 ; suma 4 par słów  (rejestr i tab C)
movaps %xmm0, (%rdx, %rax)                  ; zapis 4 słów (tab C)
addq 16,%rax                                ; przesunięcie dla słów wiersza tab B i tab C
cmpq dł_wiersza, %rax                       ; test przeliczenia wiersza tab B i tab C
jne e2
```

Kod asemblera dla zakresu 2 pętli wewnętrznych programu

```
; for k=0; k<N;k++
e1: movss (%rsi),xmm1                ;pobranie słowa z tab a
xorl %eax,%eax                       ;przesunięcie w wierszu tab b i tab c =0
shufps 0,%xmm1,%xmm1                 ;powielenie słowa z tab a

{ for j=0;j<N; j++    c[i,j]+= a[i,k]*b[k,j];      }

addq dł_wiersza, %rcx                ;adres odczytu z tab b na początku kolejnego wiersza
addq 4, %rsi                         ;adres odczytu z tab a przesunięty do kolejnego słowa
cmpq tabb+wielkośćb, %rcx            ;test przeliczenia tab b
jne e1
```

```
for k=0; k<N; k++
    …
        c[i,j]+= a[i,k]*b[k,j];
```

Kod asemblera dla zakresu 3 pętli wewnętrznych programu

```
; for i=0; i<N;i++
movl tabc, %edx                    ;adres pocz. zapisu
xorl %edi, %edi                    ;początkowy numer wiersza tab a=0
e0: movslq %edi, %rsi              ;aktualny numer wiersza tab a
movl tabb, %ecx                    ;adres pocz. tab b
imulq dł_wiersza_m, %rsi, %rsi     ;przesunięcie adresu słowa tab a
addq taba, %rsi                    ;bieżący adres słowa tab a
{ for k=0; k<N; k++
        for j=0; j<N; j++
                c[i,j]+= a[i,k]*b[k,j];          }
addl 1,%edi                        ;zwiększenie numeru wiersza tabc taba
addq liczba_bajtów_wiersza_m, %rdx ;adres początku kolejnego wiersza tabc
cmpl liczba_wierszy_m, %edi        ; test przeliczenia wszystkich wierszy
jne e0
```

```
for i=0; i<N; i++
    …
        c[i,j]+= a[i,k]*b[k,j];
```

Mnożenie macierzy kolejność pętli IKJ

- Dzięki zapewnieniu w kodzie lokalności przestrzennej dostępu do danych możliwe było efektywne zastosowanie instrukcji SSE (SEE2) pozwalających na zmniejszenie liczby dostępów do pamięci i liczby instrukcji dodawania i mnożenia.

Wielkość macierzy	N x N
Złożoność obliczeniowa	2 x N x N x N
Liczba dostępów	N x N (A) + 2* N x N x N /4 (C) + N x N x N /4 (B)
Liczba instrukcji dodawania i mnożenia (FP)	2 * N x N x N /4

Mnożenie macierzy kolejność pętli IJK

```
float a[N,N], b[N,N], c[N,N]

for i=0; i<N; i++
        for j=0;j<N; j++
                for k=0; k<N; k++
                        c[i,j]+= a[i,k]*b[k,j];
```

Wykonane jako:

- Wyznaczanie elementu macierzy wynikowej – mnożenie par słów z wiersza tablicy A i kolumny tablicy B – dostępy do słów z tablicy B nie są lokalne przestrzennie – odstęp w dostępie do danych, w efekcie brak zastosowania instrukcji przetwarzania wektorowego SSE

Liczba dostępów do danych:

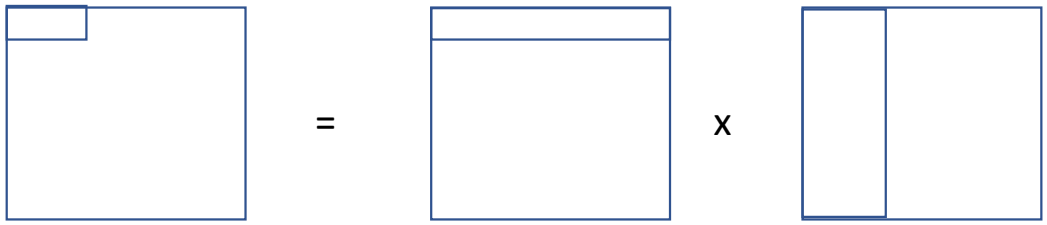
- N krotny odczyt każdego słowa tablicy a – N*N*N odczytów tablicy a
- N krotny odczyt każdego słowa tablicy b – N*N*N odczytów tablicy b
- jednokrotny odczyt i zapis każdego słowa tablicy C – 2*N*N dostępów do tablicy c

Porównanie efektywności podejść mnożenia macierzy IKJ i IJK

	IKJ	IJK
ROZMIAR MACIERZY	N*N	N*N
LICZBA DOSTĘPÓW DO PAMIĘCI	¾ N*N*N + N*N	2*N*N*N + N*N
LICZBA INSTRUKCJI ARYTMETYCZNYCH ZMIENNOPRZECINKOWYCH	½ N*N*N	2 N*N*N
RODZAJ DOSTĘPU	DOSTĘP LOKALANY PRZESTRZENNIE DO KAŻDEJ TABLICY	BRAK DOSTĘPU LOKALNEGO PRZESTRZENNIE DO TABLICY B problem z TLB
CZAS OBLICZEŃ	T	~ 10 * T (W ZALEŻNOŚCI OD WIELKOSCI INSTANCJI) RÓŻNICA PRĘDKOSCI WZRASTA Z ROZMIAREM INSTANCJI

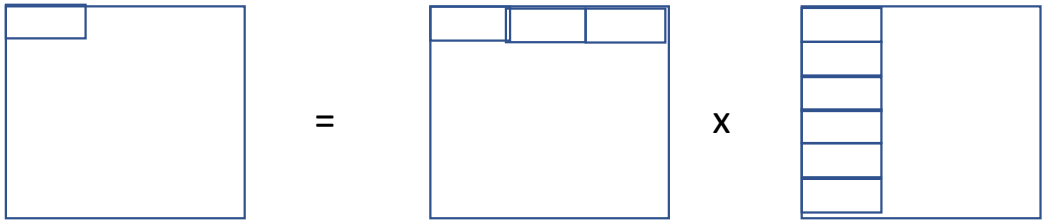
Lokalność czasowa
IJK – analiza stosunku trafień do pp

```
for ( int i = 0 ; i < n ; i++)
        for (int j = 0 ; j < n ; j++)
                for (int k = 0 ; k < n ; k++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```



Najmniejszy pojemnościowo cykl zapewniający częściową lokalność dostępu do danych:

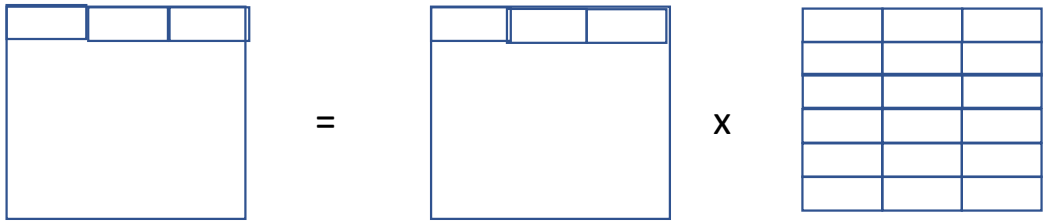
- linia ppp tablicy C
- Wiersz tablicy A
- 16 „4 bajtowych” słów tablicy B (64 B) w każdym z wierszy



Jeżeli ppp mieści zaznaczone linie pp to dane będą pobierane do pamięci podręcznej:
Tablica C – jednokrotnie
Tablica A – jednokrotnie
Tablica B – wielokrotnie, lecz raz wczytana linia pp zostanie użyta w całości i stosunek trafień do linii będzie wynosił 15/16
Tablica B - n*n*n dostępów do ppp, wszystkie linie n razy pobierane
Liczba linii wynosi sufit(4*n*n/64),
liczba braków trafień wynosi n* sufit(4*n*n/64),
Stosunek trafień to:
(n*n*n - n* sufit(4*n*n/64)) /n*n*n = ok 15/16
Jaka jest wielkość pamięci podręcznej zapewniająca taką sytuację ?

```
float c[][];
 for ( int i = 0 ; i < n ; i++)
                for (int j = 0 ; j < n ; j++)
                                for (int k = 0 ; k < n ; k++)
                                                C[i][j] + = A[i][k] * B[k][j] ;
```

IJK –
analiza stosunku trafień do pp



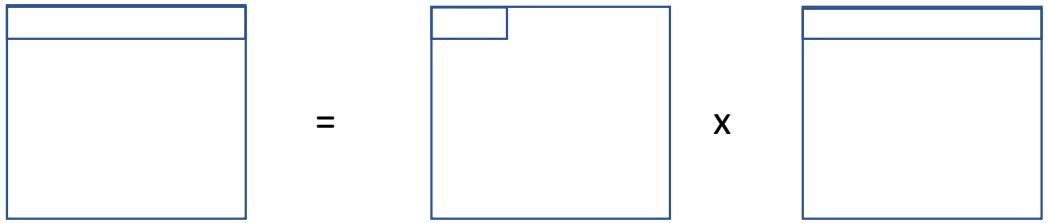
Jeżeli pp mieści zaznaczone linie pp to dane będą pobierane do pamięci jednokrotnie i stosunek trafień będzie wynikał z liczby dostępów do tablic.

Tablica C – 2*n*n dostępów, liczba braków trafień pierwszego dostępu = sufit(4*n*n/64)

Tablica A – n*n dostępów, liczba braków trafień pierwszego dostępu = sufit(4*n*n/64)

Tablica B – n*n*n dostępów, liczba braków trafień pierwszego dostępu = sufit(4*n*n/64)

```
for ( int i = 0 ; i < n ; i++)
        for (int k = 0 ; k < n ; k++)
                for (int j = 0 ; j < n ; j++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```



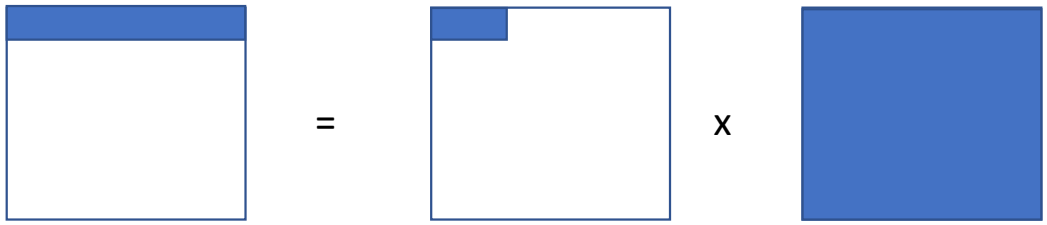
Najmniejszy pojemnościowo cykl zapewniający częściową czasową lokalność dostępu do danych:

- Wiersz tablicy C
- Linia ppp tablicy A
- Wiersz tablicy B

IKJ – analiza stosunku trafień do pp

Jeżeli pp mieści zaznaczone linie pp to dane będą pobierane do pamięci podręcznej:
Tablica C – jednokrotnie
Tablica A – jednokrotnie
Tablica B – wielokrotnie – n razy , lecz raz wczytana linia pp zostanie użyta w całości i stosunek trafień do linii będzie wynosił 15/16
Tablica B - n*n*n dostępów do ppp, wszystkie linie n razy pobierane
Liczba linii wynosi sufit(4*n*n/64),
liczba braków trafień wynosi n* sufit(4*n*n/64),
Stosunek trafień to:
(n*n*n - n* sufit(4*n*n/64)) / n*n*n = ok 15/16

```
for ( int i = 0 ; i < n ; i++)
        for (int k = 0 ; k < n ; k++)
                for (int j = 0 ; j < n ; j++)
                        C[i][j] + = A[i][k] * B[k][j] ;
```



Najmniejszy pojemnościowo cykl zapewniający czasową lokalność dostępu do danych:

- Wiersz tablicy C
- Linia ppp tablicy A
- Tablica B

IKJ – analiza stosunku trafień

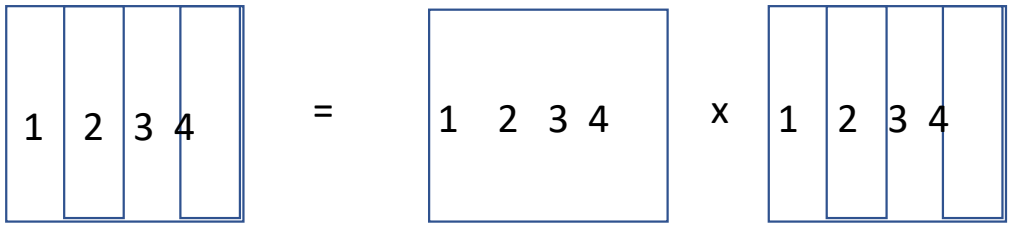
Jeżeli pp mieści zaznaczone linie pp to dane będą pobierane do pamięci podręcznej:
Tablica C – jednokrotnie
Tablica A – jednokrotnie
Tablica B – jednokrotnie
Jeżeli pp mieści zaznaczone linie pp to dane będą pobierane do pamięci jednokrotnie i stosunek trafień będzie wynikał z liczby dostępów do tablic:
C – 2*n*n*n dostępów
A – n*n dostępów
B – n*n*n dostępów

Mnożenie macierzy równolegle – dane używane przez wątki

kolejność pętli j,i,k

A,B,C są tablicami nxn

```
#pragma omp parallel for  //4 wątki podział  statyczny blokowy
 for (int j = 0 ; j < n ; j++)
    for ( int i = 0 ; i < n ; i++)
          for (int k = 0 ; k < n ; k++)
                C[i][j] + = A[i][k] * B[k][j] ;
```



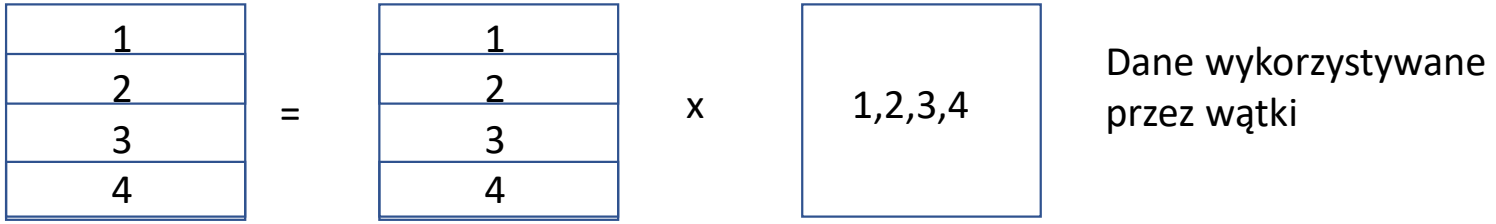
Zaznaczono dane wykorzystywane przez poszczególne wątki

C A B

Równoległe mnożenie macierzy [i,k,j]

A,B,C są tablicami nxn

```
#pragma omp parallel for // 4 wątki
 for ( int i = 0 ; i < n ; i++)
     for (int k = 0 ; k < n ; k++)
             for (int j = 0 ; j < n ; j++)
                     C[i][j] + = A[i][k] * B[k][j] ;
```

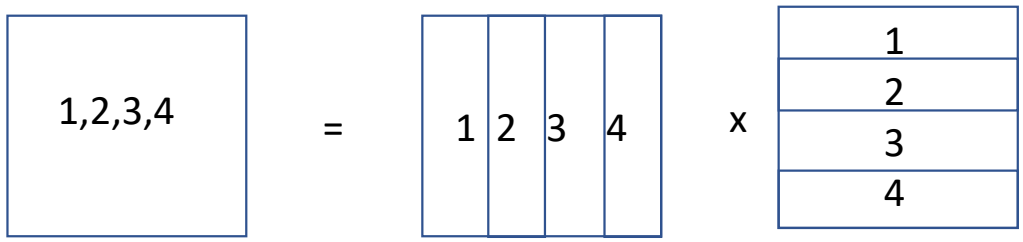


Dane wykorzystywane przez wątki

Równoległe mnożenie macierzy [i,k,j]

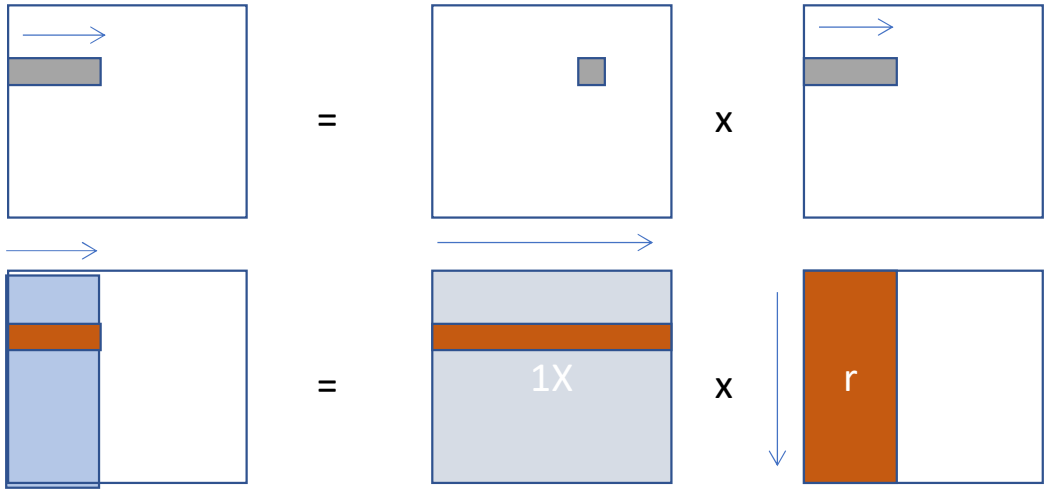
A,B,C są tablicami nxn

```
#pragma omp parallel // 4 wątki
 for ( int i = 0 ; i < n ; i++)
#pragma omp for
      for (int k = 0 ; k < n ; k++)
            for (int j = 0 ; j < n ; j++)
               C[i][j] + = A[i][k] * B[k][j] ;
```



Dane wykorzystywane przez wątki
Wyścig w dostępie do danych – kod niepoprawny

Mnożenie macierzy [i,k,j]
wiele faz obliczeń - zmniejszenie zakresu pętli wewnętrznej



zamiast:

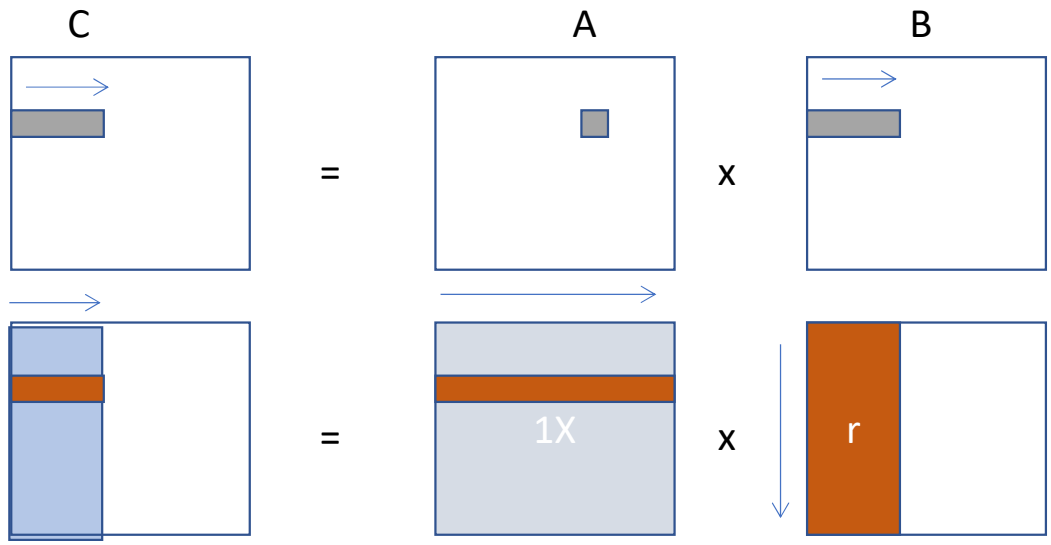
```
for ( int i = 0 ; i < n ; i++)
      for (int k = 0 ; k < n ; k++)
            for (int j = 0 ; j < n ; j++)
                  C[i][j] + = A[i][k] * B[k][j] ;
```

```
for ( int i = 0 ; i < n ; i++)
      for (int k = 0 ; k < n ; k++)
            for (int j = 0 ; j < r ; j++)
                  C[i][j] + = A[i][k] * B[k][j] ;
```

Obliczamy kolejno fragmenty kolejnych wierszy macierzy wynikowej (nie cały wiersz)

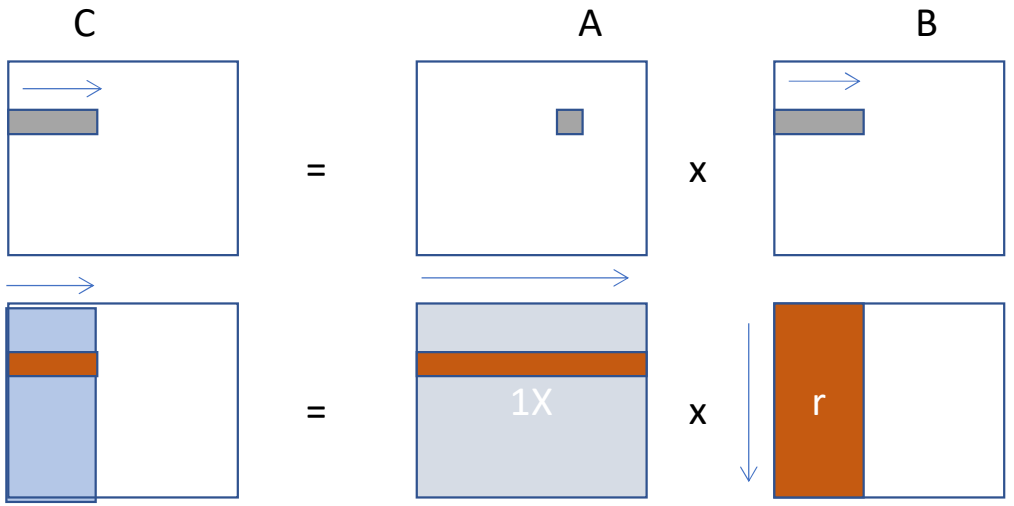
```
for ( int j = 0 ; j < n ; j+=r) // iteracje po pionowych pasach wyniku
    for ( int i = 0 ; i < n ; i++)  // wyznaczenie niebieskiej części wyniku
          for (int k = 0 ; k < n ; k++)  // wyznaczenie brązowej części wyniku
              for (int jj = j ; jj < j+r-1 ; jj++)
                    C[i][jj] + = A[i][k] * B[k][jj] ;
```

Wiele faz obliczeń - zmniejszenie zakresu pętli wewnętrznej – łatwiejsza lokalność czasowa



- Przy odpowiedniej wielkości r możliwa lokalność czasowa odwołań do danych w tablicy B - tablica B ograniczona B[*][jj:jj+r-1]
- Zmniejszenie wielkości fragmentów tablicy, na podstawie których realizowane są obliczenia prowadzi do większej lokalności odwołań.
- Konieczne ponowne pobrania macierzy A – ile razy ?
- Macierz A użyta jednokrotnie w 3 wewnętrznych pętlach (wyznaczania pasa wyniku).
- Konieczność pobrania A w każdym etapie pętli zewnętrznej (n/r razy)

Cykle pobierania danych do pp
k =N/r etapów



k=N/r razy powtórz:

- Pobranie nowej części C – 1/k *N*N słów
- Pobranie całości A
- Pobranie nowej części B - 1/k *N*N słów
- Pełne obliczenia dla pobranej części C

Wynik dla pamięci podręcznej mniejszej od N*N:
Zamiast:

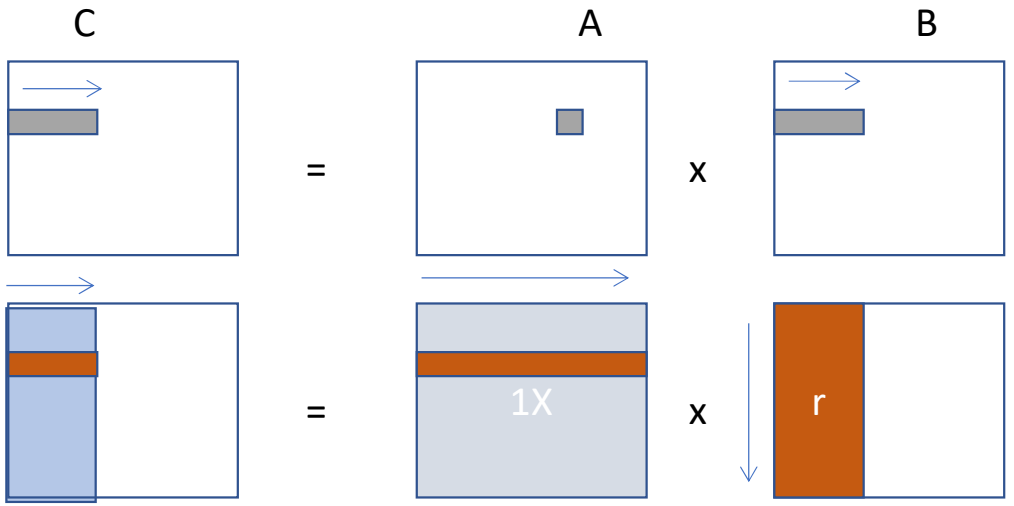
- N razy pobierać B

Tylko

- k=N/r razy pobierana tablica A

Adekwatne dla liczby pobrań stosunki trafień do ppp i prędkość przetwarzania.

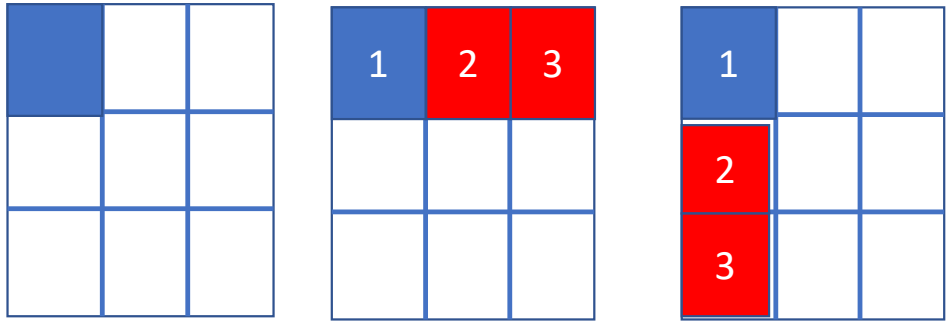
Przykład obliczeniowy parametr k metody 4 pętlowej (IKJ)



- Oznaczenia: Mnożenie tablic N*N słowo 4 bajty, pamięć podręczna o rozmiarze PP, Liczba etapów k
- Parametry: PP = 6 MB, wielkość macierzy dla N=2000 4*N*N = 16 MB
- Tablica B > PP - musi być podzielona na części
- Liczba części k > 16/6 zapewni jednoczesne przechowywanie w ppp bieżąco używanych części tablic (brązowe)
- Dla k = 3 r > 2000/3 r = 667
- Dla k = 3 zysk w ilości danych pobranych z pamięci RAM - pobrane dane to tylko 4* (2*N*N + 3*N*N) = 20 N*N zamiast 4* (2*N*N + N*N*N) w metodzie 3 pętlowej
- Dla k=3, N= 2000 słowa 4 B – 80 MB zamiast 32 GB
- Ilość danych pobranych w metodzie 4 pętlowej dla słowa W - W*N*N*(2+k) gdzie k > wielkość macierzy/ wielkość pamięci podręcznej

Wiele faz obliczeń -
zmniejszenie zakresu dwóch pętli wewnętrznych
– lepsza lokalność czasowa

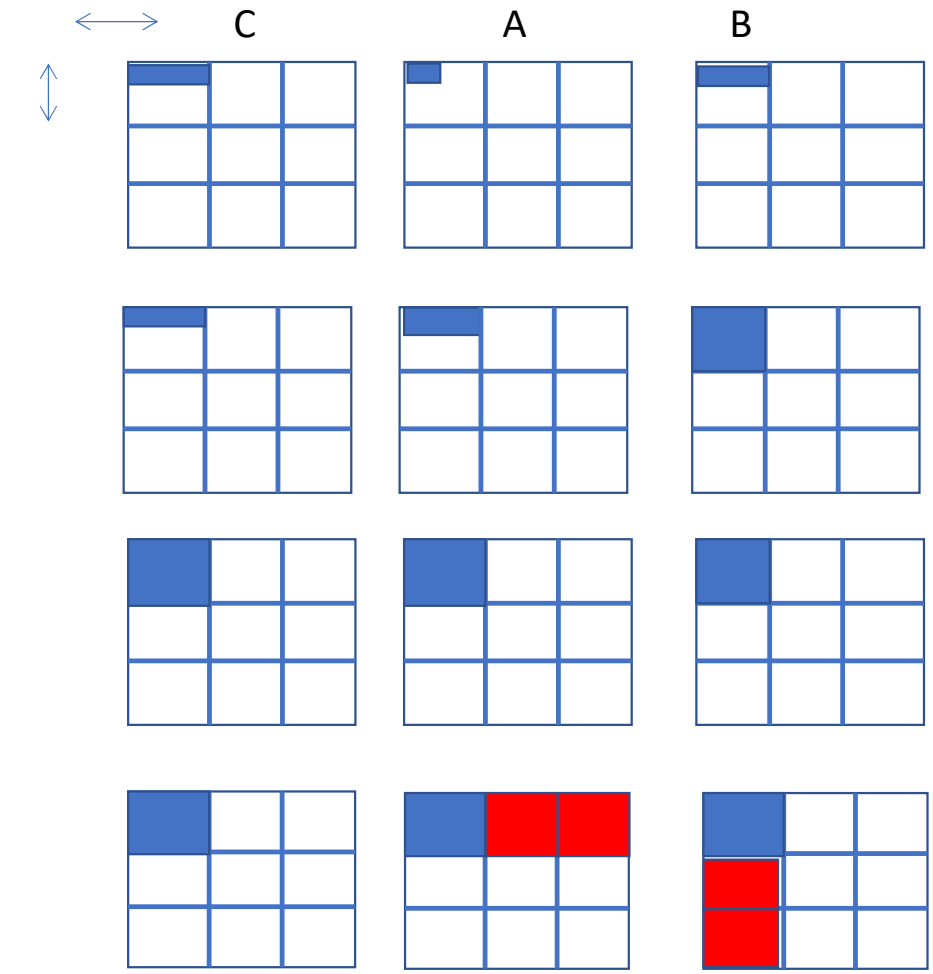
- Dwuwymiarowy podział pracy - generacja zadań pozwalających na minimalizację ilości danych używanych przez proces w każdym etapie przetwarzania.



Rysunek dla k=3 - podział macierzy na k * k części i wynik kolejno dla każdej części powstaje w k=3 etapach z ilością danych etapu = 3* r * r gdzie r=N/k wielkość bloku.

Mnożenie macierzy – dwuwymiarowy podział danych

r x r




```
for (int jj = j ; jj < j+r ; jj++)
    C[ii][jj] + = A[ii][kk] * B[kk][jj];
```

```
for (int kk = k ; kk < k+r ; kk++)
      for (int jj = j ; jj < j+r ; jj++)
            C[ii][jj] + = A[ii][kk] * B[kk][jj];
```

```
for ( int ii = i ; ii < i+r; ii++)
 for (int kk = k ; kk < k+r ; kk++)
   for (int jj = j ; jj < j+r ; jj++)
         C[ii][jj] + = A[ii][kk] * B[kk][jj];
 -- wynik częściowy
```

```
for (int k = 0 ; k < n ; k+=r)
   for ( int ii = i ; ii < i+r; ii++)
      for (int kk = k ; kk < k+r ; kk++)
         for (int jj = j ; jj < j+r ; jj++)
            C[ii][jj] + = A[ii][kk] * B[kk][jj];
-- wynik ostateczny
```

Mnożenie macierzy – podział pracy dwuwymiarowy metoda 6 pętlowa

```
for ( int i = 0 ; i < n ; i+=r) //wszystkie wiersze bloków
  for ( int j = 0 ; j < n ; j+=r) //kolejny wiersz bloków
    for (int k = 0 ; k < n ; k+=r) // wynik bloku RxR
      for ( int ii = i ; ii < i+r; ii++)//wynik częściowy blok
          for (int kk = k ; kk < k+r ; kk++)
              for (int jj = j ; jj < j+r ; jj++)
                  C[ii][jj] + = A[ii][kk] * B[kk][jj];
```

Dla C[ii][jj], A[ii][kk], B[kk][jj] lokalność czasowa dostępów do danych przy założeniu, że wszystkie podmacierze A,B i C (A[i:i+r-1][k:k+r-1],B[k:k+r-1] [j:j+r-1], C[i:i+r-1][j:j+r-1]) mieszczą się w pamięci podręcznej .

1. 3 pętle wewnętrzne służą do wyznaczenia wyniku częściowego dla fragmentu tablicy wynikowej (sum iloczynów elementów wierszy i kolumn fragmentów macierzy wejściowych),
2. czwarta pętla (po k) służy do uzupełnienia wyniku o pozostałe iloczyny wynikające z uwzględnienia kolejnych (branych po r) elementów wierszy i kolumn fragmentów macierzy wejściowych,
3. pętle piąta i szósta służą do wyznaczenia kolejnych kwadratowych (r) obszarów macierzy wynikowej.

Metoda 6 pętlowa - analiza lokalności czasowej dostępu do danych

- W ramach 4 wewnętrznych pętli wątek korzysta się z 3 tablic o wielkości r*r. Tablica wyniku jest potrzebna na każdym z kolejnych etapów, różne obszary tablic wejściowych (A i B) są kolejno potrzebne w wielkości r*r słów.
- W pamięci podręcznej o rozmiarze PP używanej przez jeden wątek zmieszczą się 3 tablice o rozmiarze

$$r <= (PP/w/3)^{1/2}$$

gdzie: w - rozmiar typu zmiennej, 3 - liczba tablic używanych przez jeden wątek

- Liczba etapów przetwarzania jest równa co najmniej $k = n/r = n / (PP/w/3)^{1/2} = (n^2*w/PP/3)^{1/2} = (3*Z)^{1/2}$

gdzie – Z = Tablica/PP jest ilorazem wielkości obliczanej macierzy kwadratowej i pamięci podręcznej komputera

- Dla przyjętej kolejności zagnieżdżenia (IKJ) podtablica B jest czytana wielokrotnie i musi być podobnie jak tablica C cały czas dostępna, podtablica A jest czytana tylko raz (wierszami) i faktycznie mogłaby (w aktualnie potrzebnym zakresie – bez potrzeby ponownego pobierania do pamięci podręcznej) zajmować jedną linię pamięci podręcznej przy efektywnym zarządzaniu pamięcią. Jednakże dla zapewnienia ciągłej obecności w pamięci wielokrotnie używanych fragmentów tablic C i B bezpieczniej również dla podtablicy A zarezerwować obszar w pamięci podręcznej równy r*r (wg wzoru powyżej).
- Iteracje zewnętrznych pętli to realizacja obliczeń powyższego typu dla innych wyników w oparciu o te same lub inne dane (tablice A i B). W całości przetwarzania każdy blok tablic A,B,C o rozmiarze r*r jest używany wielokrotnie w n/r etapach i tyle razy procesor pobiera obszary tablic A i B do pamięci podręcznej przy spełnieniu powyższej zależności na r.

Metoda 6 pętlowa równoległa - zrównoważenie obliczeń równoległych

W sytuacji, gdy przetwarzanie jest **realizowane równolegle** w zależności od sposobu podziału pracy mamy różne cechy uzyskanego przetwarzania:

- Wersja A - podział pracy przed pierwszą pętlą
  - Każdy z wątków oblicza podzbiór bloków macierzy położonych obok siebie w poziomych pasach – liczba zadań do podziału wynosi N/R i powinna być dobrana dla zapewnienia zrównoważenia pracy systemu – N/R podzielne przez liczbę procesorów (jest to dodatkowy warunek dla określenia efektywnego R).
- Wersja B - podział pracy przed pętlą czwartą
  - Każdy z wątków dzieli pracę w ramach wyliczania sum częściowych każdego wynikowego bloku c [R,R], każdy wątek liczy inną cześć wyników tablicy R*R . Liczba zadań do podziału wynosi R. Ten podział pracy wprowadza wielokrotną synchronizację wątków (wbudowana w zagnieżdżone #pragma omp for).
- Wersja C - podział pracy przed pętlą trzecią
  - wyścig w dostępie do danych – możliwe jednoczesne uaktualnienia tych samych elementów zmiennych mogą doprowadzić do błędnych wyników. Zapewnienie atomowości uaktualnienia wprowadza znaczny koszt synchronizacji wątków. Ewentualne zmienne lokalne wyników powoduję znaczny wzrost zajętości pamięci.

```
#pragma omp parallel
#pragma omp for
for ( int i = 0 ; i < n ; i+=r)
   for ( int j = 0 ; j < n ; j+=r)
//#pragma omp for
      for (int k = 0 ; k < n ; k+=r)
//#pragma omp for
         for ( int ii = i ; ii < i+r; ii
            for (int kk = k ; kk < k+r ; kk++)
               for (int jj = j ; jj < j+r ; jj++)
                  C[ii][jj] + = A[ii][kk] * B[kk][jj];
```

Porównanie metod mnożenia macierzy (IKJ)

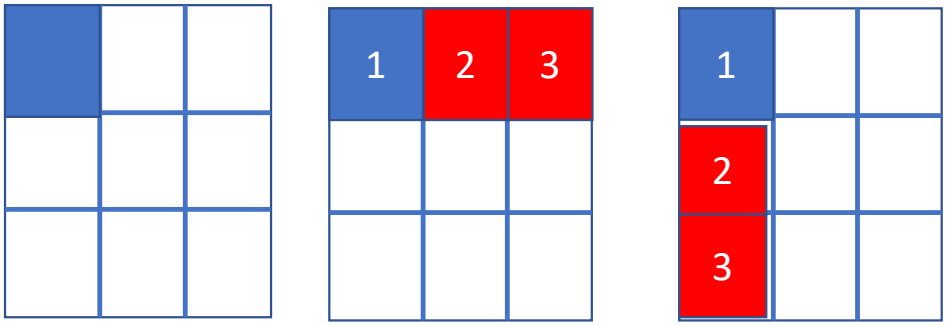
Wzory na liczbę etapów i ilość danych pobieranych z RAM

Liczba pętli	liczba etapów pracy	Ilość słów pobranych z RAM
3	1	2*N*N+ N*N*N
4	Tablica/PP	N*N*(2+k)
6	$(3*\text{Tablica}/PP)^{1/2}$	N*N*(1+2*k)
Tablica = liczba bajtów tablicy	PP = liczba bajtów pp	N = liczba wierszy/kolumn

Wzory dotyczą sytuacji gdy:

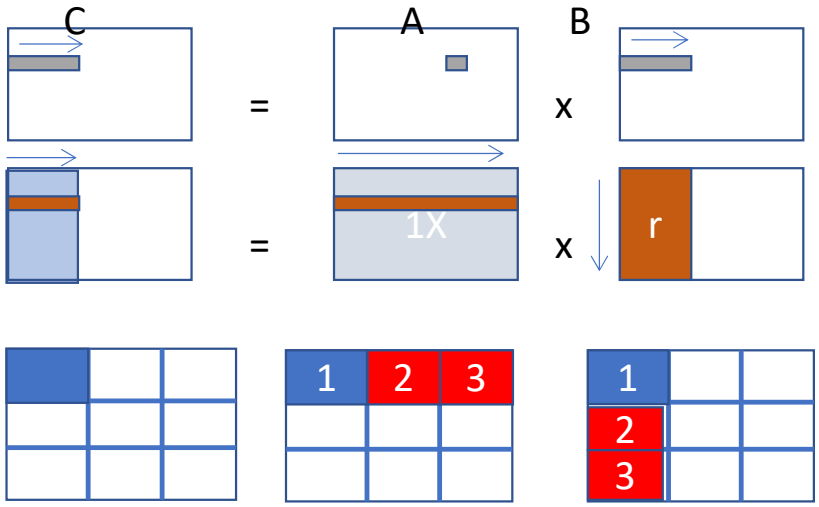
- Tablica > PP i liczba etapów lokalnego dostępu do danych k > 1
- 3*N*w < PP – pojedyncze wiersze A, B, C mieszczą się w PP

Przykład obliczeniowy parametr metody 6 pętlowej



- Oznaczenia: Mnożenie tablic N*N słowo 4 bajty (w), pamięć podręczna o rozmiarze PP, liczba etapów k
- Parametry: PP = 12 MB, wielkość macierzy dla N=3000 4*N*N = 36 MB
- Tablica B > PP dlatego tablice A,B,C muszą być podzielone na części o rozmiarze R < $(PP/(4*3))^{1/2} = 1000$
- Liczba etapów k >= N/R = 3 zapewni etapową lokalność czasową dostępów
- Dla k = 3 R = 1000
- Dla k = 3 zysk w ilości danych pobranych z RAM - pobrane dane to tylko 4* (N*N + 2*k*N*N) = 28 * N * N zamiast 4* (2*N*N + N*N*N) w metodzie 3 pętlowej
- Zysk w ilości danych pobranych z pamięci RAM do PP dla k=3, N= 3000 słowa 4 B – 252 MB zamiast 216 GB

Przykład obliczeniowy porównanie metod 4 i 6 pętlowej (IKJ) dla sytuacji braku lokalności czasowej dostępu do danych PP < MACIERZ



- Oznaczenia: Mnożenie tablic N*N słowo 4 bajty, pamięć podręczna o rozmiarze PP, liczba etapów k
- 4 pętle
- Parametry: PP = 3 MB, wielkość macierzy dla N=5000 4*N*N = 100 MB
- k > 100/3 k = 34
- dla k = 34 r ≥ 5000/34 r = 148
- Dla k = 34, N= 5000
- Ilość danych pobranych z pamięci RAM dla metody 4 pętlowej 3,6 GB w*N*N*(2+k) = 4*25*36 MB
- 6 pętli
- R ≤ $(PP/(4*3))^{1/2} = 500$
- k ≥ N/R k = 10
- Dla k = 10 R = 500
- Ilość danych pobranych z pamięci RAM dla metody 6 pętlowej 2,1 GB
- w*N*N*(1+2*k) = 4*25*(1+20) MB

Dla PP 3 MB i N = 5000
Dane pobrane z pamięci RAM:

- 3 pętle – 500 GB
- 4 pętle – 3,6 GB
- 6 pętli – 2,1 GB

Metoda 6 pętlowa - analiza lokalności przestrzennej

Lokalność przestrzenna dostępu do danych wynika:

- z postaci najbardziej wewnętrznej pętli programu (czytanie tablicy B wierszami),
- z wartości parametru r (rozmiar 2D bloku danych).

Zastosowany podział przetwarzania na etapy o mniejszej ilości danych powoduje, że lokalność przestrzenna wynikająca z kolejności zagnieżdżenia pętli pozostaje.