**1. Wstęp**

W przeprowadzanym przez nas eksperymencie badaliśmy efektywność trzech metod mnożenia macierzy:

* metody 3-pętlowej IJK
* metody 6-pętlowej IJK ii-jj-kk
* metody 3-pętlowej IKJ (wykonywanej sekwencyjnie jako pomiar będący punktem odniesienia do wyżej wymienionych badań przeprowadzanych współbieżnie)

Mnożone są dwie macierze kwadratowe o rozmiarze **n**.

Wynikowe elementy obliczane są w następujący sposób:

ci,j =

Eksperyment był przeprowadzany z wykorzystaniem procesora **AMD Phenom II X4 945** o następujących parametrach:

* architektura: 64-bitowa
* liczba rdzeni: 4
* liczba wątków: 4
* częstotliwość taktowania: 3 GHz
* pamięć cache:

- L1 : 4x64 KB

- L2: 4x512 KB

- L3: 6 MB

**2. Analiza algorytmów**

**2.1. Implementacja**

a). metoda 3-pętlowa IJK

void multiply\_matrices\_IJK() {

#pragma omp parallel for

for (int i = 0 ; i < ROWS ; i++)

for (int j = 0 ; j < COLUMNS ; j++)

for (int k = 0 ; k < COLUMNS ; k++)

matrix\_r[i][j] += matrix\_a[i][k] \* matrix\_b[k][j] ;

}

b). metoda 6-pętlowa IJK ii-jj-kk

void multiply\_matrices\_IJK\_IJK(int R){  
 #pragma omp parallel for  
 for (int i = 0; i < ROWS; i+=R)   
 for (int j = 0; j < COLUMNS; j+=R)   
 for (int k = 0; k < COLUMNS; k+=R)   
 for (int ii = i; ii < i + R; ii++)   
 for (int jj = j; jj < j + R; jj++)   
 for (int kk = k; kk < k + R; kk++)   
 matrix\_r[ii][jj] += matrix\_a[ii][kk] \* matrix\_b[kk][jj];  
}

c). metoda 3-pętlowa IKJ (sekwencyjna)

void multiply\_matrices\_IKJ\_seq() {

for (int i = 0 ; i < ROWS ; i++)

for (int k = 0 ; k < COLUMNS ; k++)

for (int j = 0 ; j < COLUMNS ; j++)

matrix\_r[i][j] += matrix\_a[i][k] \* matrix\_b[k][j] ;

}

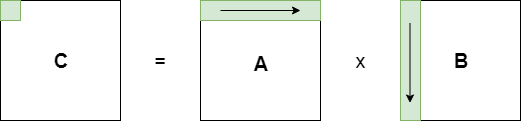
**2.2. Opis algorytmów**

W metodzie **3-pętlowej** najbardziej wewnętrzna pętla (K) iteruje po kolumnach macierzy A oraz po wierszach macierzy B w celu uzyskania pojedynczego elementu macierzy C.  
Pętle I oraz J wskazują, który element macierzy wynikowej jest aktualnie liczony - zewnętrzna pętla I określa jej numer wiersza natomiast J kolumnę.   
Złożoność obliczeniowa tego algorytmu to *O(n3)*.

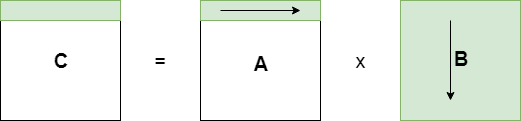
W metodzie **6-pętlowej** macierz dzielona jest na bloki o rozmiarze R.  
Elementy wynikowe liczone są dla poszczególnych bloków analogicznie do metody 3-pętlowej dla całej macierzy.   
Złożoność algorytmu się nie zmienia ( O(n3) ), jednak takie podejście może mieć wpływ na lokalność czasową dostępu do pamięci, co wykażemy w dalszej części sprawozdania.

**2.3. Zastosowanie dyrektyw Open MP**

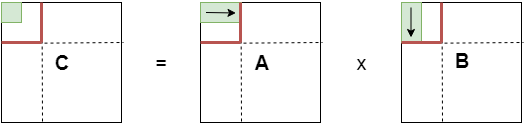
Zarówno dla metody 3-pętlowej jak i 6-pętlowej użyliśmy dyrektywy **#pragma omp parallel for** przed pierwszą pętlą. Powoduje ona rozdzielenie operacji równomiernie pomiędzy wszystkie pracujące wątki. Oznacza to, że każdy z czterech dostępnych wątków otrzyma równy ( w miarę możliwości ) przydział iteracji do wykonania. Każda z nich będzie wykonana dokładnie jeden raz (równolegle z innymi).



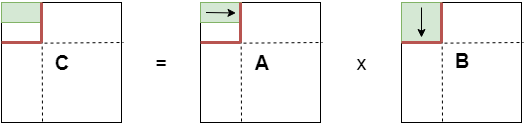
Rysunek 1: Zasoby wykorzystywane w pętli wewnętrznej w metody 3-pętlowej



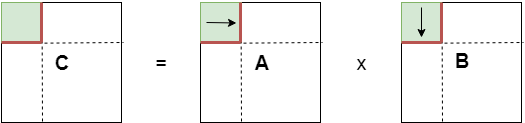
Rysunek 2: Zasoby wykorzystywane w pętli środkowej metody 3-pętlowej



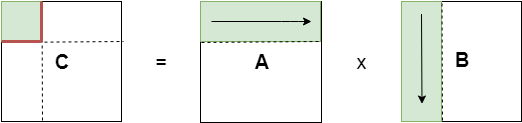
Rysunek 3: Zasoby wykorzystywane w pętli szóstej metody 6-pętlowej



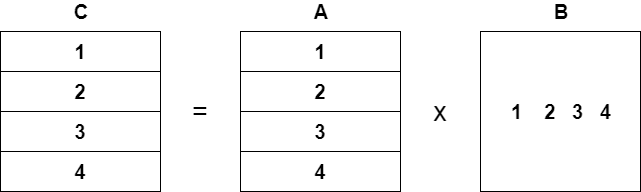
Rysunek 4: Zasoby wykorzystywane w pętli piątej metody 6-pętlowej



Rysunek 5: Zasoby wykorzystywane w pętli czwartej metody 6-pętlowej



Rysunek 6: Zasoby wykorzystywane w pętli trzeciej metody 6-pętlowej



Rysunek 7: Podział pracy i zasobów pomiędzy poszczególne wątki