**Przetwarzanie równoległe   
Open MP**

**Autorzy:**

Paweł Kuffel gr. I1

Jakub Wąsik 132335 gr. I1 ( [jakub.wasik@student.put.poznan.pl](mailto:jakub.wasik@student.put.poznan.pl) )

Laboratorium: wtorek, 15:10, tygodnie nieparzyste

**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. Obliczenie bloku macierzy wynikowej o rozmiarze R następuje w wyniku wykonania się całej trzeciej pętli.  
Złożoność algorytmu się nie zmienia ( *O(n3)* ), jednak takie podejście może mieć korzystny wpływ na lokalność czasową dostępu do pamięci, co wykażemy w dalszej części sprawozdania.

**2.3. Zastosowanie dyrektyw Open MP**

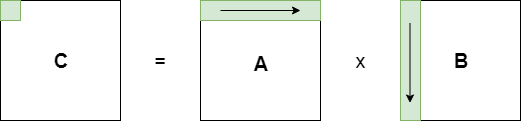
Zarówno dla metody 3-pętlowej jak i 6-pętlowej użyliśmy dyrektywy **#pragma omp parallel for** przed pierwszą pętlą. Powoduje ona rozdzielenie operacji równomiernie pomiędzy wszystkie pracujące wątki. Oznacza to, że każdy z czterech dostępnych wątków otrzyma równy ( w miarę możliwości ) przydział iteracji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

**2.4. Podział pracy i zasobów między wątki**

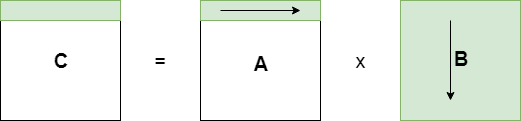
a). metoda 3-pętlowa IJK

W celu obliczenia pojedynczego elementu macierzy wynikowej (co dzieje się podczas wykonania pętli wewnętrznej) algorytm iteruje jednokrotnie po kolumnach macierzy A oraz wierszach macierzy B.

Jedno wykonanie pętli środkowej to obliczenie całego wiersza macierzy wynikowej.   
W tym celu wielokrotnie będziemy odwoływać się do jednego wiersza macierzy A oraz przejdziemy jednokrotnie każdą kolumnę macierzy B.



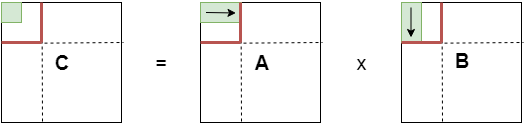
Rysunek 1: Zasoby wykorzystywane w pętli wewnętrznej w metody 3-pętlowej



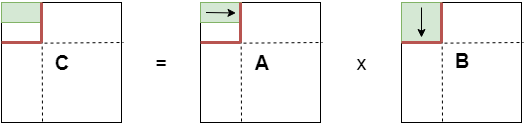
Rysunek 2: Zasoby wykorzystywane w pętli środkowej metody 3-pętlowej

a). metoda 6-pętlowa IJK ii-jj-kk

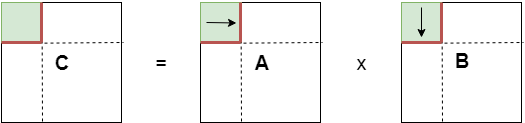
Jak wcześniej wspomnieliśmy, w trakcie pracy tego algorytmu wyznaczany jest kwadratowy obszar macierzy C o rozmiarze R.   
Dokonuje się to poprzez mnożenie fragmentów macierzy A i B o tym samym rozmiarze.  
Trzy najbardziej wewnętrzne pętle wykonują analogiczną pracę jak w przypadku algorytmu   
3-pętlowego, tylko dla podmacierzy o rozmiarze R.  
Aby wyznaczony fragment macierzy wynikowej był kompletny, operacje te należy powtórzyć dla kolejnych fragmentów macierzy A oraz B, przesuwając się o odpowiednio R kolumn i R wierszy, co następuje podczas wykonania pętli trzeciej.



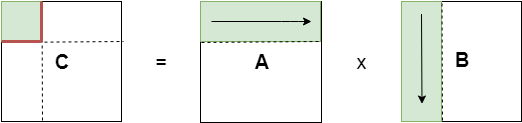
Rysunek 3: Zasoby wykorzystywane w pętli szóstej metody 6-pętlowej



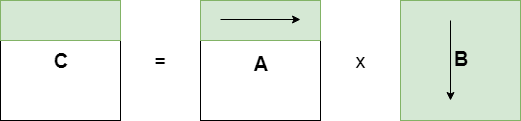
Rysunek 4: Zasoby wykorzystywane w pętli piątej metody 6-pętlowej



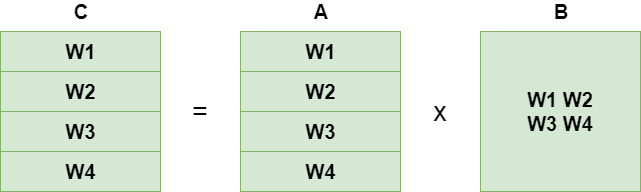
Rysunek 5: Zasoby wykorzystywane w pętli czwartej metody 6-pętlowej



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

  
Rysunek 7: Zasoby wykorzystywane w pętlach wewnętrznych metody 6-pętlowej

Takie wykonywanie mnożenia powoduje, że praca może zostać podzielona **w ten sam sposób dla obu metod**. Każdy z wątków czyta przydzieloną mu część wierszy macierzy A (rozłączną od pozostałych) oraz całą macierz B, aby obliczyć konkretny zestaw wierszy macierzy wynikowej C.



Rysunek 8: Podział pracy i zasobów pomiędzy poszczególne wątki

**2.5. Potencjalne problemy**

a). **wyścig** – **nie występuje.** Każdy fragment macierzy wynikowej C przetwarzany jest przez osobny wątek, zatem nie ma możliwości nadpisania danej czytanej/zapisanej przez inny wątek.   
Elementy macierzy A oraz B są danymi tylko do odczytu, więc w przypadku operacji na tych zbiorach danych wyścig w ogóle nie zachodzi.

b) **false sharing** – **nie występuje**. Dzięki podziałowi pracy przed pętlą zewnętrzną, każdy z procesorów otrzymuje do zapisu rozłączne podzbiory macierzy wynikowej (macierze A i B nie grają w tym przypadku roli, ponieważ następuje na nich wyłącznie odczyt).  
Tylko niewielka część danych (z końca przedziału zasobów przydzielonych poszczególnym wątkom) może wchodzić do tej samej linii pamięci w różnych procesorach, jednak nie jest to znaczące dla efektywności przetwarzania.   
Oznacza to, że nie wystąpi sytuacja, w której wątki będą nadpisywały dane, pochodzące z tej samej linii pamięci (której szczególnym przypadkiem jest właśnie **false sharing –** nadpisywanie różnych danych, z tej samej linii, skutkujące koniecznością uaktualnienia jej w drugim procesorze).

**3. Analiza lokalności**

**3.1. Lokalność czasowa**

W celu zapewnienia wysokiej efektywności przetwarzania, należy troszczyć się o to, aby dane raz ładowane do pamięci podręcznej były użyte wielokrotnie, zanim zostaną usunięte czy unieważnione. Taki stan rzeczy nazywamy **czasową lokalnością odwołań.**  
Jej brak może skutkować niskim stosunkiem trafień do pamięci podręcznej i spowolnienie przetwarzania.

Aby zapewnić lokalność czasową, musimy wziąć pod uwagę rozmiar pamięci L3 (współdzielonej pomiędzy poszczególnymi wątkami) i dopasować do niej wielkość instancji macierzy.   
Dalsze rozważania opieramy o wcześniejszą analizę zasobów potrzebnych do wykonania wewnętrznych pętli poszczególnych algorytmów.  
  
W celu jak najmniejszej liczby odwołań do pamięci, w metodzie **3-pętlowej** w 6MB pamięci L3 musimy zmieścić:

* całą macierz B
* cztery wiersze macierzy A (po jednym liczonym przez każdy dostępny wątek)
* cztery elementy macierzy wynikowej C (po jednym liczonym przez każdy dostępny wątek)

Zmienne zawarte w macierzach są typu **float -** ich rozmiar wynosi 4B.

Rozmiar instancji N musi zatem spełniać warunek:

Oznacza to, że N w przybliżeniu byłoby równe 1250.

W przypadku metody **6-pętlowej** lokalność czasową powinno zapewnić przechowanie w pamięci podręcznej trzech macierzy kwadratowych o rozmiarze R dla każdego wątku.   
R musi zatem spełniać warunek:

Jako że praca jest przydzielana wątkom równomiernie przed rozpoczęciem najbardziej zewnętrznej pętli, to iloraz **N/R** powinien być **wielokrotnością czwórki**, aby system pracował w sposób zrównoważony.

**3.2. Lokalność przestrzenna**

przestrzenna lokalność odwołań

korzystanie w kodzie z danych zajmujących sąsiednie lokacje w pamięci

– brak plo powoduje niski stosunek trafień do pp i do bufora translacji

adresów.

(np. jeżeli tablica jest zapisywana wierszami w pamięci to kolejne dostępy do tablicy

powinny też, jeśli to możliwe, być realizowane wierszami).