Szczecin, 22.10.2023

Miłosz Misiek

Nr albumu: 54764

**Automatyczna Optymalizacja Oprogramowania**

Laboratorium nr 3

**Cel:** *„Znajdowanie zależności z wykorzystaniem PET i kalkulatora ISCC; generowanie kodu; implementacja postprocesora, opracowanie programu do porównania wyników produkowanych przez program źródłowy i docelowy”*

**Wstęp**

**Parametry komputera:**

|  |  |
| --- | --- |
| **Czip** | Apple M1 Pro |
| **Łączna liczba rdzeni** | 8 |
| **OS** | macOS Sonoma 14.0.0 |
| **Pamięć cache** | **L1:** 192k+128k x6, 128k+64k x2  **L2/L3:** 12 MB x6, 4 MB x2 |
| **Pamięć operacyjna** | 16 GB |

Wariant do opracowania: **3**

1. **Pętla**

for(i=1;i<=n;i++)

for(j=3;j<=n;j++)

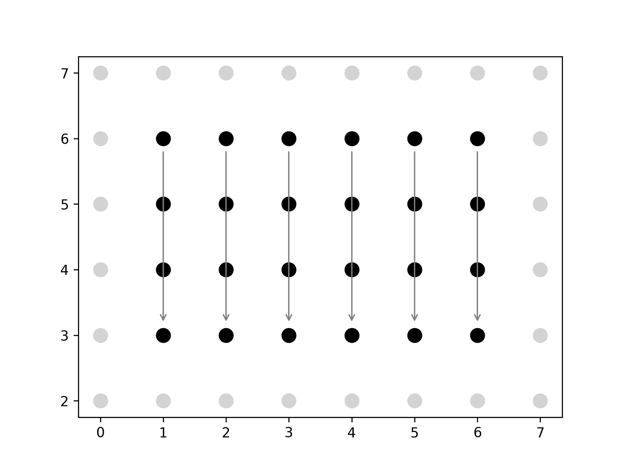
a[i][j] = a[i][j-3];

**Zadania**

1. **Dla wskazanej pętli za pomocą kalkulatora ISCC znaleźć relację zależności, R, przestrzeń iteracji, LD oraz zrobić rysunek grafu zależności w przestrzeni 6 x 6.**

R:=[n]-> {[i, j] -> [i, j+3] : 1 <= i <= n and 3 <= j <= n};

LD:=[n]-> {[i, j]: 1 <= i <= n and 3 <= j <= n};



1. **Za pomocą operatora kalkulatora ISCC: IslSchedule := schedule LD respecting R minimizing R znaleźć szeregowanie afiniczne w postaci drzewa**

R:=[n]-> {[i, j] -> [i, j+3] : 1 <= i <= n and 3 <= j <= n};

LD:=[n]-> {[i, j]: 1 <= i <= n and 3 <= j <= n};

IslSchedule := schedule LD respecting R minimizing R;

print "IslSchedule";

IslSchedule; "IslSchedule"

$0 := { domain: "[n] -> { [i, j] : 0 < i <= n and 3 <= j <= n }", child: { schedule: "[n] -> [{ [i, j] -> [(i)] }, { [i, j] -> [(j)] }]", permutable: 1, coincident: [ 1, 1 ] } }

**Interpretacja:**

schedule: "[n] -> [{ [i, j] -> [(i)] }, { [i, j] -> [(j)] }]"

* Pierwsza transformacja to **{ [i, j] -> [ ( i ) ] }**, która ustala harmonogram oparty na wymiarze **i**
* Druga transformacja to **{ [i, j] -> [ ( j ) ] },** która ustala harmonogram oparty na wymiarze j.

Te transformacje określają kolejność wykonywania instancji instrukcji zdefiniowanych w dziedzinie.

1. **Za pomocą operatora kalkulatora ISCC: map przekonwertować szeregowanie afiniczne w postaci drzewa na szeregowanie w postaci relacji**

SCHED:=map IslSchedule;

print "SCHED";

"SCHED"

SCHED;

$2 := [n] -> { [i, j] -> [i, j] }

1. **Sprawdzić które (tylko jedno) z uzyskanych szeregowań pozwala na ekstrakcję równoległości pozbawionej synchronizacji. Stosując właściwe szeregowanie wygenerować pseudokod i kod kompilowalny reprezentujący równoległość pozbawioną synchronizacji.**

D:=deltas R;

print "D";

"D"

D;

$3 := [n] -> { [i = 0, j = 3] : n >= 3 }

1. **{ [i, j] -> [(i)] }** - z wartościami **(c11, c12) = (1, 0)**
2. **{ [i, j] -> [(j)] }** - z wartościami **(c21, c22) = (0, 1)**

Zbiór **D** reprezentuje wektor dystansu: **[i = 0, j = 3]**, więc: **(d11, d12) = (0, 3)**

Sprawdzanie pierwszego szeregowania: c11*d11 + c12*d12 = 1*0 + 0*3 = 0.

Nie ma drugiego wektora dystansu, więc nie trzeba sprawdzać drugiego równania dla pierwszego szeregowania.

Sprawdzanie drugiego szeregowania jest zbędne, ponieważ pierwsze szeregowanie już spełnia wymagania.

CODE\_SYNCH\_FREE:=[n] -> { [i, j] -> [p,i,j] :p=i }\*LD;

CODE\_SYNCH\_FREE;

$4 := [n] -> { [i, j] -> [p = i, i, j] : 0 < i <= n and 3 <= j <= n }

1. **Stosując uzyskaną relację CODE\_SYNCH\_FREE za pomocą operatora scan znaleźć wszystkie niezależne fragmenty kodu i zaznaczyć je na rysunku stworzonym w p. 1 (rysunek z zależnościami) w przestrzeni 6x6 (12x12 jeśli przestrzeń została rozszerzona, patrz p.1).**

print "scan (CODE\_SYNCH\_FREE\* [n]->{:n=6})";

"scan (CODE\_SYNCH\_FREE\* [n]->{:n=6})"

scan (CODE\_SYNCH\_FREE\* [n]->{:n=6});

$7 := [n] -> { [i = 6, j = 6] -> [p = 6, 6, 6] : n = 6; [i = 5, j = 6] -> [p = 5, 5, 6] : n = 6; [i = 4, j = 6] -> [p = 4, 4, 6] : n = 6; [i = 3, j = 6] -> [p = 3, 3, 6] : n = 6; [i = 2, j = 6] -> [p = 2, 2, 6] : n = 6; [i = 1, j = 6] -> [p = 1, 1, 6] : n = 6; [i = 6, j = 5] -> [p = 6, 6, 5] : n = 6; [i = 5, j = 5] -> [p = 5, 5, 5] : n = 6; [i = 4, j = 5] -> [p = 4, 4, 5] : n = 6; [i = 3, j = 5] -> [p = 3, 3, 5] : n = 6; [i = 2, j = 5] -> [p = 2, 2, 5] : n = 6; [i = 1, j = 5] -> [p = 1, 1, 5] : n = 6; [i = 6, j = 4] -> [p = 6, 6, 4] : n = 6; [i = 5, j = 4] -> [p = 5, 5, 4] : n = 6; [i = 4, j = 4] -> [p = 4, 4, 4] : n = 6; [i = 3, j = 4] -> [p = 3, 3, 4] : n = 6; [i = 2, j = 4] -> [p = 2, 2, 4] : n = 6; [i = 1, j = 4] -> [p = 1, 1, 4] : n = 6; [i = 6, j = 3] -> [p = 6, 6, 3] : n = 6; [i = 5, j = 3] -> [p = 5, 5, 3] : n = 6; [i = 4, j = 3] -> [p = 4, 4, 3] : n = 6; [i = 3, j = 3] -> [p = 3, 3, 3] : n = 6; [i = 2, j = 3] -> [p = 2, 2, 3] : n = 6; [i = 1, j = 3] -> [p = 1, 1, 3] : n = 6 }

print "CODE\_SYNCH\_FREE";

"CODE\_SYNCH\_FREE"

CODE\_SYNCH\_FREE;

$8 := [n] -> { [i, j] -> [p = i, i, j] : 0 < i <= n and 3 <= j <= n }

A diagram of a graph

Description automatically generated

1. **Wygenerować pseudokod i kod kompilowalny implementujący równoległość pozbawioną synchronizacji.**

* pseudokod

codegen CODE\_SYNCH\_FREE;

for (int c0 = 1; c0 <= n; c0 += 1)

for (int c2 = 3; c2 <= n; c2 += 1)

(c0, c2);

* kod preprocesora kompilowany

// zad\_6.c

#include <stdio.h>

#include <omp.h>

int main()

{

const int Rows = 7;

const int Columns = 8; // dodatkowa kolumna żeby nie wyjść poza obszar przy petli.

int a[Rows][Columns];

// wypelnienie macierzy przykładowymi liczbami.

int counter = 1;

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

{

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

{

a[i][j] = counter++;

}

}

#pragma omp parallel for

for (int c0 = 1; c0 <= 6; c0++)

{

for (int c2 = 3; c2 <= 6; c2++)

{

a[c0][c2] = a[c0][c2 + 1];

}

}

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 6; j++)

{

printf("%d ", a[i][j]);

}

printf("\n");

}

return 0;

}

* clang -Xpreprocessor -fopenmp ./zad\_6.c -o zad\_6\_wynik -lomp
* ./zad\_6\_wynik > zad\_6\_wynik.txt

**Wynik**1 2 3 4 5 6 **(pierwszy wiersz niezmieniony, ponieważ pętla zaczyna się od c0 = 1)**

9 10 11 13 14 15 **(12 został zastąpiony przez 13, 13 przez 14, itd.)**

17 18 19 21 22 23

25 26 27 29 30 31

33 34 35 37 38 39

41 42 43 45 46 47

Każda liczba w tablicy od drugiego wiersza w górę i od czwartej kolumny w prawo została przesunięta w lewo, a na jej miejsce weszła wartość z prawej strony.

1. **Zastosować program do porównania produkowanych przez pętle wyników (zadanie z   
   poprzedniego laboratorium) do sprawdzenia poprawności kodu docelowego w przestrzeni   
   6x6.**

* Porównanie wyników

Wynik został porównany z wynikiem uzyskanym przez kompilacje kodu sekwencyjnego. Kod jest praktycznie tożsamy, usunięto tylko bibliotekę OMP i dyrektywę paralel for dla pętli.

clang compare.c -o compare

./compare kod\_docelowy\_wynik. kod\_docelowy\_seq\_.txt

**Uwaga:** nic nie zostało zwrócone -> wyniki są tożsame

**Wnioski**

* Za pomocą kalkulatora ISCC możliwe było znalezienie relacji zależności oraz przestrzeni iteracji dla danej pętli.
* Wykorzystując operator kalkulatora ISCC, przekształcono szeregowanie afiniczne z formy drzewa do formy relacji, co pozwoliło na bardziej zrozumiałą interpretację.
* Zastosowanie operatorów matematycznych na relacjach pozwoliło zidentyfikować, które z szeregowań umożliwiają ekstrakcję równoległości pozbawionej synchronizacji.
* Analiza uzyskanej relacji CODE\_SYNCH\_FREE umożliwiła identyfikację niezależnych fragmentów kodu.
* Generacja pseudokodu oraz kodu kompilowalnego była prosta i bezpośrednia, co świadczy o skuteczności użytych metod optymalizacji.
* Równoległe wykorzystanie pętli w kodzie kompilowalnym, z wykorzystaniem OpenMP, zapewnia efektywną realizację zadania w środowisku wielowątkowym.