Laboratorium 3: Zastosowanie szeregowania afinicznego do znalezienia równoległości pozbawionej synchronizacji

Wariant pętli 2

```
for(i=1; i<=n; i++)
    for(j=2; j<=n; j++)
        a[i][j] = a[i][j-2];
```

Zadanie 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.

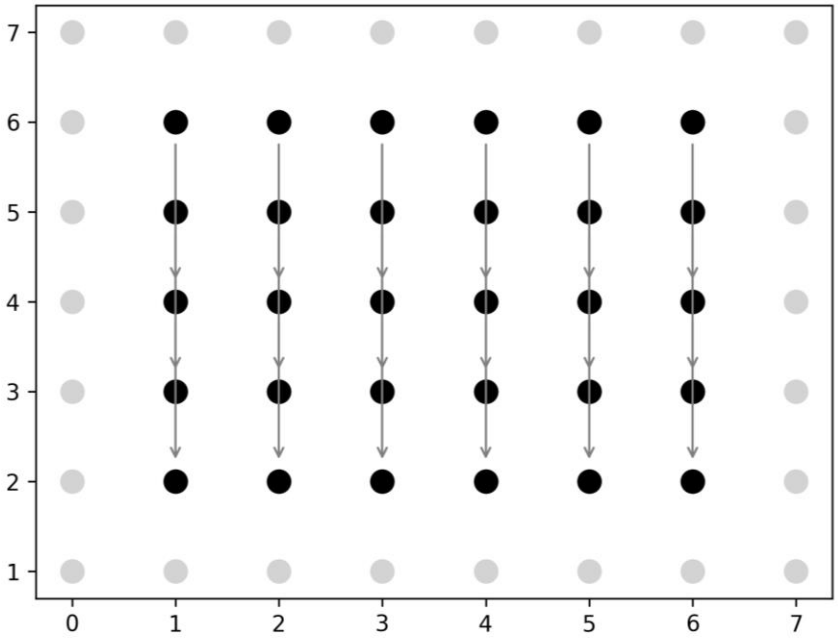
Relacja R:

[n] -> { [i, j] -> [i1, j1] : i > 0 and 2 <= j <= n and i < i1 <= n and 2 <= j1 <= n; [i, j] -> [i1 = i, j1] : 0 < i <= n and j >= 2 and j < j1 <= n; [i, j] -> [i1 = i, j1 = 2 + j] : 0 < i < n and 2 <= j <= -2 + n; [i, j] -> [i1] : 0 < i <= n and 2 <= j <= n and i1 > i and 0 < i1 <= n; [i] -> [i1, j] : i > 0 and i < i1 <= n and 2 <= j <= n; [i] -> [i1 = i, j] : 0 < i <= n and 2 <= j <= n; [i] -> [i1] : i > 0 and i < i1 <= n; [] -> [i] : 0 < i <= n }

Przestrzeń iteracji LD (Loop Domain):

[n] -> { []; [i] : 0 < i <= n; [i, j] : 0 < i <= n and 2 <= j <= n }

Rysunek grafu w przestrzeni 6x6:



Zadanie 2.

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

Szeregowanie afiniczne w postaci drzewa:

domain: "[n] -> { []; [i] : 0 < i <= n; [i, j] : 0 < i <= n and 2 <= j <= n }"

child:

```
 schedule: "[n] -> [{ [] -> [(0)]; [i] -> [(i)]; [i, j] -> [(i)] }]"
permutable: 1
 coincident: [ 1 ]
 child:
  set:
  - filter: "[n] -> { [i]; [i, j] }"
    child:
     schedule: "[n] -> [{ [i] -> [(0)]; [i, j] -> [(j)] }]"
     permutable: 1
     coincident: [ 1 ]
  - filter: "[n] -> { [] }"
```

W powyższym drzewie znajdują się dwa następujące szeregowania ISL:

- [n] -> [{ [] -> [(0)]; [i] -> [(i)]; [i, j] -> [(i)] }]
- [n] -> [{ [i] -> [(0)]; [i, j] -> [(j)] }]

Zadanie 3.

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

Szeregowanie w postaci relacji:
[n] -> { [i, j] -> [i, 0, j]; [] -> [0, 1, 0]; [i] -> [i, 0, 0] }

Zadanie 4.

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.

Wektory dystansu:
[n] -> { [i, j] : 0 < i < n and 2 - n <= j <= -2 + n; [i = 0, j] : 0 < j <= -2 + n; [i = 0, j = 2] : n >= 4; [i] : 0 < i < n }

Do wyboru odpowiedniego szeregowania do wygenerowania kodu bez synchronizacji będą używane następujące szeregowania:

- [n] -> [{ [] -> [(0)]; [i] -> [(i)]; **[i, j] -> [(i)]** }], czyli **(1, 0)**
- [n] -> [{ [i] -> [(0)]; **[i, j] -> [(j)]** }], czyli **(0, 1)**

i następujący wektor dystansu: (0, 2).

Żeby sprawdzić, czy jest równoległość pozbawiona synchronizacji, sprawdzamy najpierw pierwsze szeregowanie:
(1, 0) * (0, 2)
szeregowanie 1 * wektor dystansu

$c_{11} * d_{11} + c_{12} * d_{22} = 1 * 0 + 0 * 2 = 0$

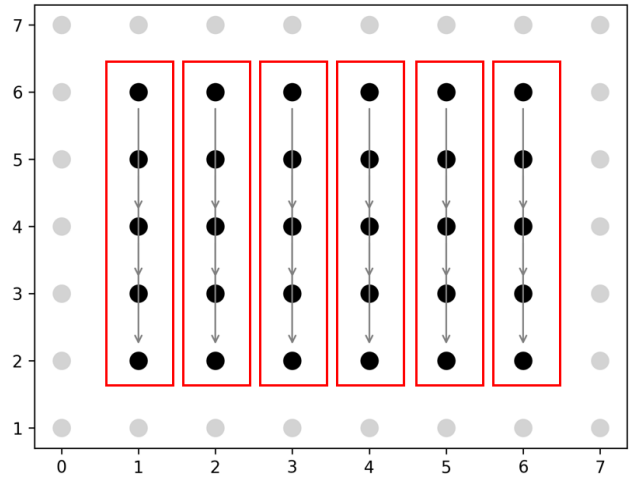
Ze względu na to, że wynik równania wynosi 0, to do wygenerowania kodu bez synchronizacji używane będzie szeregowanie pierwsze, czyli [i, j] -> [(i)] lub inaczej (1, 0) .

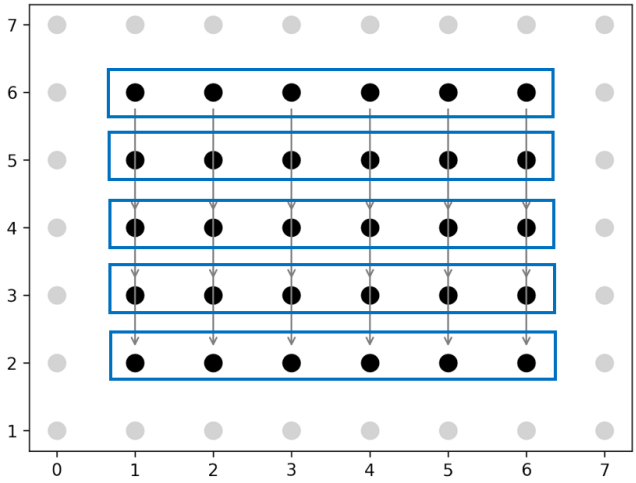
Zadanie 5.

Stosując uzyskaną relację CODE_SYNCH_FREE za pomocą operatora scan znaleźć wszystkie niezależne fragmenty kodu i zaznaczyć je na rysunku stworzonym w p. 1 w przestrzeni 6x6.

Niezależne fragmenty kodu, znalezione przy pomocy operatora *scan*:
[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; [i = 6, j = 2] -> [p = 6, 6, 2] : n = 6; [i = 5, j = 2] -> [p = 5, 5, 2] : n = 6; [i = 4, j = 2] -> [p = 4, 4, 2] : n = 6; [i = 3, j = 2] -> [p = 3, 3, 2] : n = 6; [i = 2, j = 2] -> [p = 2, 2, 2] : n = 6; [i = 1, j = 2] -> [p = 1, 1, 2] : n = 6 }

Na poniższym rysunku na osi x znajdują się indeksy procesorów, a na osi y - indeksy iteracji. Wynika z tego, że w tym samym czasie mogą być wykonywane wszystkie operacje wewnątrz jednej iteracji (zaznaczone na niebiesko), np wszystkie operacje dla iteracji nr 2. Na czerwono zaznaczono iteracje, które będą wykonywane na tym samym procesorze.





Zadanie 6.
Wygenerować pseudokod i kod kompilowalny implementujący równoległość pozbawioną synchronizacji.

Wygenerowany pseudokod:

```
for (int c0 = 1; c0 <= n; c0 += 1)
  for (int c2 = 2; c2 <= n; c2 += 1)
    (c0, c2);
```

Kod komplilowalny:

```
for (int c0 = 1; c0 <= n; c0 += 1)
  for (int c2 = 2; c2 <= n; c2 += 1)
    a[c0][c2] = a[c0][c2+1];
```

Zadanie 7.
Zastosować program do porównania produkowanych przez pętle wyników do sprawdzenia poprawności kodu docelowego w przestrzeni 6x6.

```
// gcc -fopenmp 5-compare.c && ./a.out

#include <stdio.h>
#include<time.h>

int main() {
    int n = 6;
    int aParallel[n+1][n+1];
    int aGenerated[n+1][n+1];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            aParallel[i][j] = j;
            aGenerated[i][j] = j;
    }

    // wejściowy
    for (int i = 1; i <= n; i++) {
        for (int j = 2; j <= n; j++) {
            aParallel[i][j] = aParallel[i][j-2];
        }
    }

    // wygenerowany
    #pragma openmp parallel for
    for (int c0 = 1; c0 <= n; c0 += 1)
        for (int c2 = 2; c2 <= n; c2 += 1) {
            aGenerated[c0][c2] = aGenerated[c0][c2-2];
    }

    printf("Parallel code result:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%02d ", aParallel[i][j]);
        }
        printf("\n");
    }

    printf("\nGenerated code result:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%02d ", aGenerated[i][j]);
        }
        printf("\n");
    }


    int noMatchCount = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (aParallel[i][j] != aGenerated[i][j]) {
                noMatchCount++;
            }
```

```
        }
    }

    if (noMatchCount > 0)
        printf("\nResults are not identical. %d values do not match.\n", noMatchCount);
    else
        printf("\nResults are identical.\n");

    return 0;
}
```

Wynik powyższego programu:

```
Parallel code result:
00 01 02 03 04 05
00 01 00 01 00 01
00 01 00 01 00 01
00 01 00 01 00 01
00 01 00 01 00 01
00 01 00 01 00 01

Generated code result:
00 01 02 03 04 05
00 01 00 01 00 01
00 01 00 01 00 01
00 01 00 01 00 01
00 01 00 01 00 01
00 01 00 01 00 01

Results are identical.
```

Załączniki.

Skrypt implementujący zadania.

```
#krok 1
#relacja zaleznosci
#R:=[n]-> {[i, j] -> [i, j+1] : 1 <= i <= n and 1 <= j < n};
P := parse_file "1-my.c";

R:=[n]-> { [i, j] -> [i1] : 0 < i <= n and 2 <= j <= n and i1 > i and 0 < i1 <= n; [i, j] -> [i1,
j1] : 0 < i <= n and 2 <= j <= n and i1 > i and 0 < i1 <= n and 2 <= j1 <= n; [i, j] -> [i1 = i,
j1] : 0 < i <= n and 2 <= j <= n and j1 > j and 2 <= j1 <= n; [i] -> [i1] : 0 < i <= n and i1 > i
and 0 < i1 <= n; [] -> [i] : 0 < i <= n; [i, j] -> [i1 = i, j1 = 2 + j] : 0 < i < n and 2 <= j <=
-2 + n; [i] -> [i1, j] : 0 < i <= n and i1 > i and 0 < i1 <= n and 2 <= j <= n; [i] -> [i1 = i, j]
: 0 < i <= n and 2 <= j <= n; [i] -> [i1] : 0 < i <= n and i1 > i and 0 < i1 <= n };
print "R";
R;

LD:=[n] -> { [i, j] : 0 < i <= n and 2 <= j <= n; []; [i] : 0 < i <= n; [i] : 0 < i <= n; [i, j] :
0 < i <= n and 2 <= j <= n };
print "LD";
```

```
LD;

print "scan (R*[n]->{:n=6})";
scan (R*[n]->{:n=6});

##krok 2
IslSchedule := schedule LD respecting R minimizing R;
print "IslSchedule";
IslSchedule;

##krok 3
SCHED:=map IslSchedule;
print "SCHED";
SCHED;

## krok 4
## obliczamy wektory dystansu:
D:=deltas R;
print "D";
D;

CHECK1:=[n]-> {[I]: I=1*0 + 0*1 };
print "CHECK1";
CHECK1;

CODE_SYNCH_FREE:=[n] -> { [i, j] -> [p,i,j] : p=i }*LD;

#krok 5
print "scan (CODE_SYNCH_FREE* [n]->{:n=6})";
scan (CODE_SYNCH_FREE* [n]->{:n=6});
print "CODE_SYNCH_FREE";
CODE_SYNCH_FREE;

#krok 6
##generujemy pseudokod
codegen CODE_SYNCH_FREE;
```