Laboratorium 5: Zastosowanie szeregowania afinicznego do implementacji techniki blokowania pętli

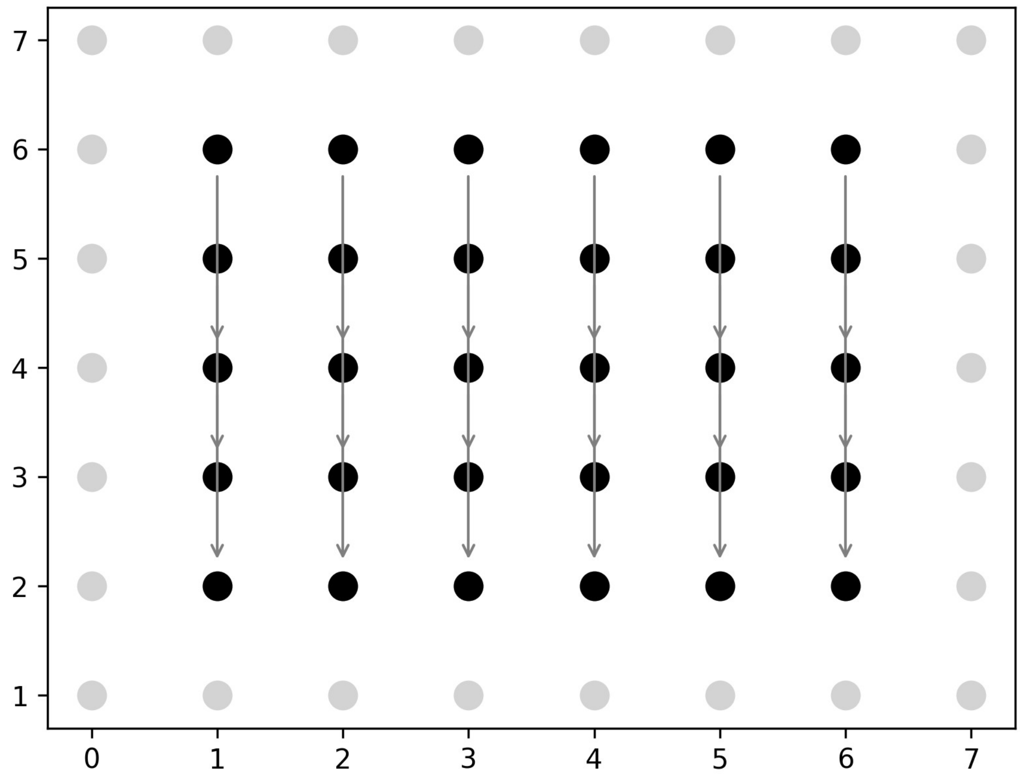
Wariant pętli 2
```
for(i=1; i<=n; i++)
    for(j=2; j<=n; j++)
        a[i][j] = a[i][j-2];
```

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] -> { S_0[i, j] -> S_0[i' = i, j' = 2 + j] : 0 < i < n and 2 <= j <= -2 + n }
```

Przestrzeń iteracji LD (Loop Domain):
```
[n] -> { S_0[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] -> { S_0[i, j] : 0 < i <= n and 2 <= j <= n }"
```

```
child:
  schedule: "[n] -> [{ S_0[i, j] -> [(i)] }, { S_0[i, j] -> [(j)] }]"
  permutable: 1
  coincident: [ 1, 1 ]
```

W powyższym drzewie znajdują się dwa następujące szeregowania ISL:

- {[i, j] -> [(i)] }
- {[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] -> { S_0[i, j] -> [i, j] }
```

Zadanie 4.

Utworzyć szeregowanie, które pozwala na implementację techniki fali frontowej na poziomie iteracji (wave-fronting).

```
"WAVE_FR"
[n] -> { S_0[i, j] -> [i + j, j] : 0 < i <= n and 2 <= j <= n }
```

Zadanie 5.

Utworzyć szeregowanie, które pozwala na implementację techniki kafelkowania (tiling) z rozmiarem kafelka 2x2, sekwencyjny sposób wykonywania kafelków.

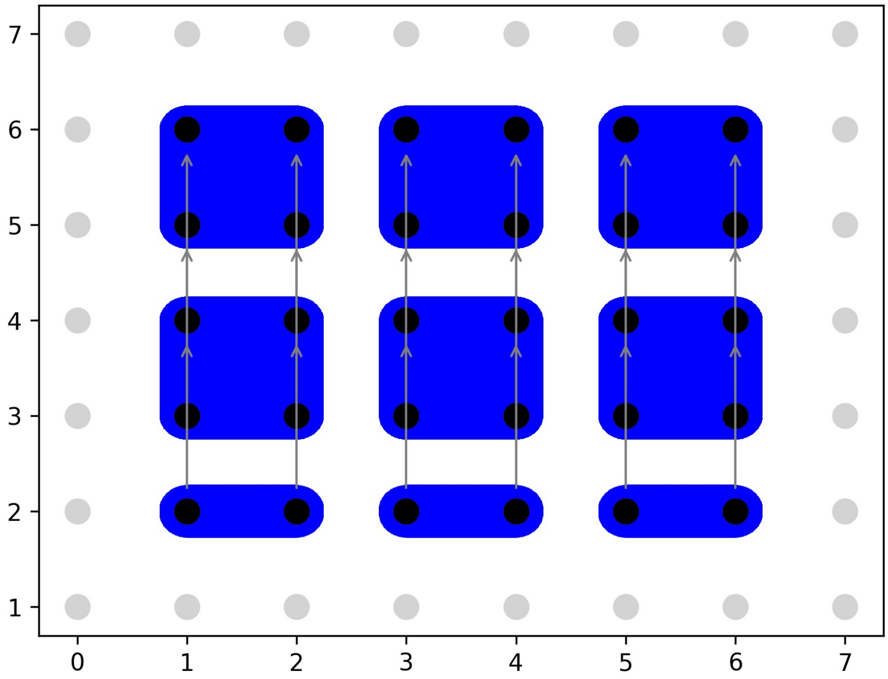
```
"CODE_TILING"
[n] -> { [i, j] -> [it, jt, i, j] : 0 < i <= n and 2 <= j <= n and -2 + i <= 2it < i and
-2 + j <= 2jt < j }
```

Zadanie 6.

Stosując uzyskane w punkcie 5 szeregowanie za pomocą operatora scan znaleźć wszystkie kafelki w przestrzeni 6x6 (12x12) i zaznaczyć je na rysunku utworzonego w p.1.

```
[n] -> { [i = 6, j = 6] -> [it = 2, jt = 2, 6, 6] : n = 6; [i = 5, j = 6] -> [it = 2, jt
= 2, 5, 6] : n = 6; [i = 4, j = 6] -> [it = 1, jt = 2, 4, 6] : n = 6; [i = 3, j = 6] ->
[it = 1, jt = 2, 3, 6] : n = 6; [i = 2, j = 6] -> [it = 0, jt = 2, 2, 6] : n = 6; [i = 1,
j = 6] -> [it = 0, jt = 2, 1, 6] : n = 6; [i = 6, j = 5] -> [it = 2, jt = 2, 6, 5] : n =
6; [i = 5, j = 5] -> [it = 2, jt = 2, 5, 5] : n = 6; [i = 4, j = 5] -> [it = 1, jt = 2,
4, 5] : n = 6; [i = 3, j = 5] -> [it = 1, jt = 2, 3, 5] : n = 6; [i = 2, j = 5] -> [it =
0, jt = 2, 2, 5] : n = 6; [i = 1, j = 5] -> [it = 0, jt = 2, 1, 5] : n = 6; [i = 6, j =
4] -> [it = 2, jt = 1, 6, 4] : n = 6; [i = 5, j = 4] -> [it = 2, jt = 1, 5, 4] : n = 6;
[i = 4, j = 4] -> [it = 1, jt = 1, 4, 4] : n = 6; [i = 3, j = 4] -> [it = 1, jt = 1, 3,
4] : n = 6; [i = 2, j = 4] -> [it = 0, jt = 1, 2, 4] : n = 6; [i = 1, j = 4] -> [it = 0,
jt = 1, 1, 4] : n = 6; [i = 6, j = 3] -> [it = 2, jt = 1, 6, 3] : n = 6; [i = 5, j = 3]
-> [it = 2, jt = 1, 5, 3] : n = 6; [i = 4, j = 3] -> [it = 1, jt = 1, 4, 3] : n = 6; [i =
3, j = 3] -> [it = 1, jt = 1, 3, 3] : n = 6; [i = 2, j = 3] -> [it = 0, jt = 1, 2, 3] : n
= 6; [i = 1, j = 3] -> [it = 0, jt = 1, 1, 3] : n = 6; [i = 6, j = 2] -> [it = 2, jt = 0,
6, 2] : n = 6; [i = 5, j = 2] -> [it = 2, jt = 0, 5, 2] : n = 6; [i = 4, j = 2] -> [it =
```

```
1, jt = 0, 4, 2] : n = 6; [i = 3, j = 2] -> [it = 1, jt = 0, 3, 2] : n = 6; [i = 2, j = 2] -> [it = 0, jt = 0, 2, 2] : n = 6; [i = 1, j = 2] -> [it = 0, jt = 0, 1, 2] : n = 6 }
```



Zadanie 7.

Wygenerować pseudokod sekwencyjny i odpowiedni kod kompilowany (sekwencyjny) implementujący kafelkowanie.

Wygenerowany pseudokod sekwencyjny:

```
for (int c0 = 0; c0 < floord(n + 1, 2); c0 += 1)
  for (int c1 = 0; c1 < (n + 1) / 2; c1 += 1)
    for (int c2 = 2 * c0 + 1; c2 <= min(n, 2 * c0 + 2); c2 += 1)
      for (int c3 = max(2, 2 * c1 + 1); c3 <= min(n, 2 * c1 + 2); c3 += 1)
        (c2, c3);
```

Kod kompilowalny:

```
for (int c0 = 0; c0 < floord(n + 1, 2); c0 += 1)
  #pragma openmp parallel for
  for (int c1 = 0; c1 < (n + 1) / 2; c1 += 1)
    for (int c2 = 2 * c0 + 1; c2 <= min(n, 2 * c0 + 2); c2 += 1)
      for (int c3 = max(2, 2 * c1 + 1); c3 <= min(n, 2 * c1 + 2); c3 += 1)
        a[c2][c3] = a[c2][c3-2];
```

Zadanie 8.

Zastosować program porównujący wyniki obliczeń (zadanie 7, L2) do sprawdzania poprawności kodu docelowego wygenerowanego w p.6 w przestrzeni 6x6.

```
Initial code result:
00 01 02 03 04 05 06
00 01 00 01 00 01 00
00 01 00 01 00 01 00
```

```
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00

Generated code result:
00 01 02 03 04 05 06
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
```

Zadanie 9.

Utworzyć szeregowanie, które pozwala na implementację techniki wave-fronting na poziomie kafli (tiles) z rozmiarem kafelka 2x2, równoległy sposób wykonywania kafelków.

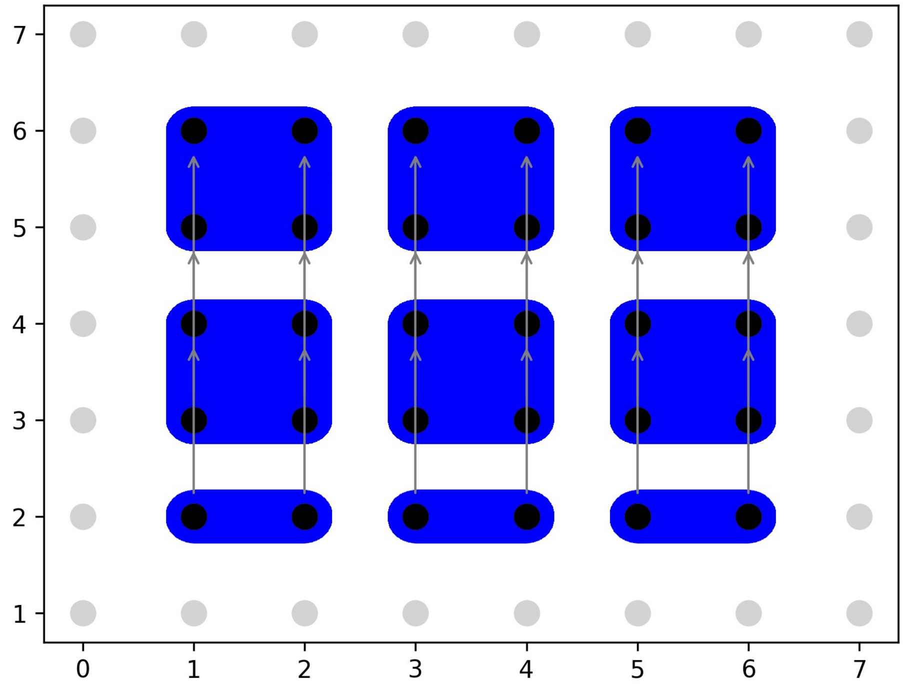
"CODE_TILING_PAR"

```
[n] -> { [i, j] -> [it, jt, i, j] : 0 < i <= n and 0 < j <= n and 2jt >= -2 + j and -i + 2it < 2jt <= 2 - i + 2it and 2jt < j }
```

Zadanie 10.

Stosując uzyskane w punkcie 9 szeregowanie za pomocą operatora scan znaleźć wszystkie partycje czasu na poziomie kafelków w przestrzeni 6x6 (12x12) i zaznaczyć je na rysunku utworzonego w p.1. Czasem wykonania kafelka jest wartość iteratora it.

```
"scan (CODE_TILING*[n]->{:n=6})"
[n] -> { [i = 6, j = 6] -> [it = 4, jt = 2, 6, 6] : n = 6; [i = 5, j = 6] -> [it = 4, jt = 2, 5, 6] : n = 6; [i = 4, j = 6] -> [it = 3, jt = 2, 4, 6] : n = 6; [i = 3, j = 6] -> [it = 3, jt = 2, 3, 6] : n = 6; [i = 2, j = 6] -> [it = 2, jt = 2, 2, 6] : n = 6; [i = 1, j = 6] -> [it = 2, jt = 2, 1, 6] : n = 6; [i = 6, j = 5] -> [it = 4, jt = 2, 6, 5] : n = 6; [i = 5, j = 5] -> [it = 4, jt = 2, 5, 5] : n = 6; [i = 4, j = 5] -> [it = 3, jt = 2, 4, 5] : n = 6; [i = 3, j = 5] -> [it = 3, jt = 2, 3, 5] : n = 6; [i = 2, j = 5] -> [it = 2, jt = 2, 2, 5] : n = 6; [i = 1, j = 5] -> [it = 2, jt = 2, 1, 5] : n = 6; [i = 6, j = 4] -> [it = 3, jt = 1, 6, 4] : n = 6; [i = 5, j = 4] -> [it = 3, jt = 1, 5, 4] : n = 6; [i = 4, j = 4] -> [it = 2, jt = 1, 4, 4] : n = 6; [i = 3, j = 4] -> [it = 2, jt = 1, 3, 4] : n = 6; [i = 2, j = 4] -> [it = 1, jt = 1, 2, 4] : n = 6; [i = 1, j = 4] -> [it = 1, jt = 1, 1, 4] : n = 6; [i = 6, j = 3] -> [it = 3, jt = 1, 6, 3] : n = 6; [i = 5, j = 3] -> [it = 3, jt = 1, 5, 3] : n = 6; [i = 4, j = 3] -> [it = 2, jt = 1, 4, 3] : n = 6; [i = 3, j = 3] -> [it = 2, jt = 1, 3, 3] : n = 6; [i = 2, j = 3] -> [it = 1, jt = 1, 2, 3] : n = 6; [i = 1, j = 3] -> [it = 1, jt = 1, 1, 3] : n = 6; [i = 6, j = 2] -> [it = 2, jt = 0, 6, 2] : n = 6; [i = 5, j = 2] -> [it = 2, jt = 0, 5, 2] : n = 6; [i = 4, j = 2] -> [it = 1, jt = 0, 4, 2] : n = 6; [i = 3, j = 2] -> [it = 1, jt = 0, 3, 2] : n = 6; [i = 2, j = 2] -> [it = 0, jt = 0, 2, 2] : n = 6; [i = 1, j = 2] -> [it = 0, jt = 0, 1, 2] : n = 6; [i = 6, j = 1] -> [it = 2, jt = 0, 6, 1] : n = 6; [i = 5, j = 1] -> [it = 2, jt = 0, 5, 1] : n = 6; [i = 4, j = 1] -> [it = 1, jt = 0, 4, 1] : n = 6; [i = 3, j = 1] -> [it = 1, jt = 0, 3, 1] : n = 6; [i = 2, j = 1] -> [it = 0, jt = 0, 2, 1] : n = 6; [i = 1, j = 1] -> [it = 0, jt = 0, 1, 1] : n = 6 }
```



Zadanie 11.
Wygenerować pseudokod i równoległy kod kompilowalny implementujący kafelkowanie.

Wygenerowany pseudokod sekwencyjny:
```
for (int c0 = 0; c0 < floord(n + 1, 2); c0 += 1)
  for (int c1 = 0; c1 < (n + 1) / 2; c1 += 1)
    for (int c2 = 2 * c0 + 1; c2 <= min(n, 2 * c0 + 2); c2 += 1)
      for (int c3 = max(2, 2 * c1 + 1); c3 <= min(n, 2 * c1 + 2); c3 += 1)
        (c2, c3);
```

Kod kompilowany:
```
for (int c0 = 0; c0 < floord(n + 1, 2); c0 += 1)
  #pragma openmp parallel for
  for (int c1 = 0; c1 < (n + 1) / 2; c1 += 1)
    for (int c2 = 2 * c0 + 1; c2 <= min(n, 2 * c0 + 2); c2 += 1)
      for (int c3 = max(2, 2 * c1 + 1); c3 <= min(n, 2 * c1 + 2); c3 += 1)
        a[c2][c3] = a[c2][c3-2];
```

Zadanie 12.
Zastosować program porównujący wyniki obliczeń (zadanie 7, L2) do sprawdzania poprawności kodu docelowego w przestrzeni 6x6.

```
# gcc -fopenmp 12-joined.c -lm && ./a.out
Initial code result:
00 01 02 03 04 05 06
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
```

```
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00

Generated code result:
00 01 02 03 04 05 06
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00
00 01 00 01 00 01 00

Results are identical.
```

Załączniki.

Skrypt implementujący zadania.

```
##Znalezienie zaleznosci
P := parse_file "1-my.c";

print "Loop domain:";
Domain := P[0];
LD := Domain;
print Domain;

Write := P[1] * Domain;
Read := P[3] * Domain;
Schedule := P[4];
Before := Schedule << Schedule;
RaW := (Write . (Read^-1)) * Before;
WaW := (Write . (Write^-1)) * Before;
WaR := (Read . (Write^-1)) * Before;

R := (RaW+WaW+WaR);
print "R";
print R;

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
WAVE_FR:=[n] -> { S_0[i, j] -> [i+j, j] }*LD;
print "WAVE_FR";
```

```
WAVE_FR;

##krok 5, kafelkowanie, generowanie kodu sekwencyjnego
CODE_TILING:= [n]->{
    [i,j]->[it, jt, i, j]:
    0 < i <= n and
    1 < j <= n and
    1<= i-2it <=2 and
    1<= j-2jt <=2
};

print "CODE_TILING";
CODE_TILING;

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

##krok 7 generowanie kodu sekwencyjnego
print "codegen CODE_TILING";
codegen CODE_TILING;

##krok 9 tworzenie relacji pozwalajacej na implementacje wave-
#fronting na poziomie kafelkow:
CODE_TILING_PAR:= [n]->{
    [i,j]->[it, jt, i, j]:
        0 < i <= n and
        1 < j <= n and
        1<= i-2it <=2 and
        1<= j-2jt <=2
};

print  "CODE_TILING_PAR";
CODE_TILING_PAR;

#krok 11 generowanie kodu rownoleglego
codegen CODE_TILING_PAR;
```