**Laboratorium 4: Zastosowanie szeregowania afinicznego do znalezienia partycjonowania czasu.**

**Wariant pętli 1**

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

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

a[i][j] = a[i][j-1] + a[i+1][j];

**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.**

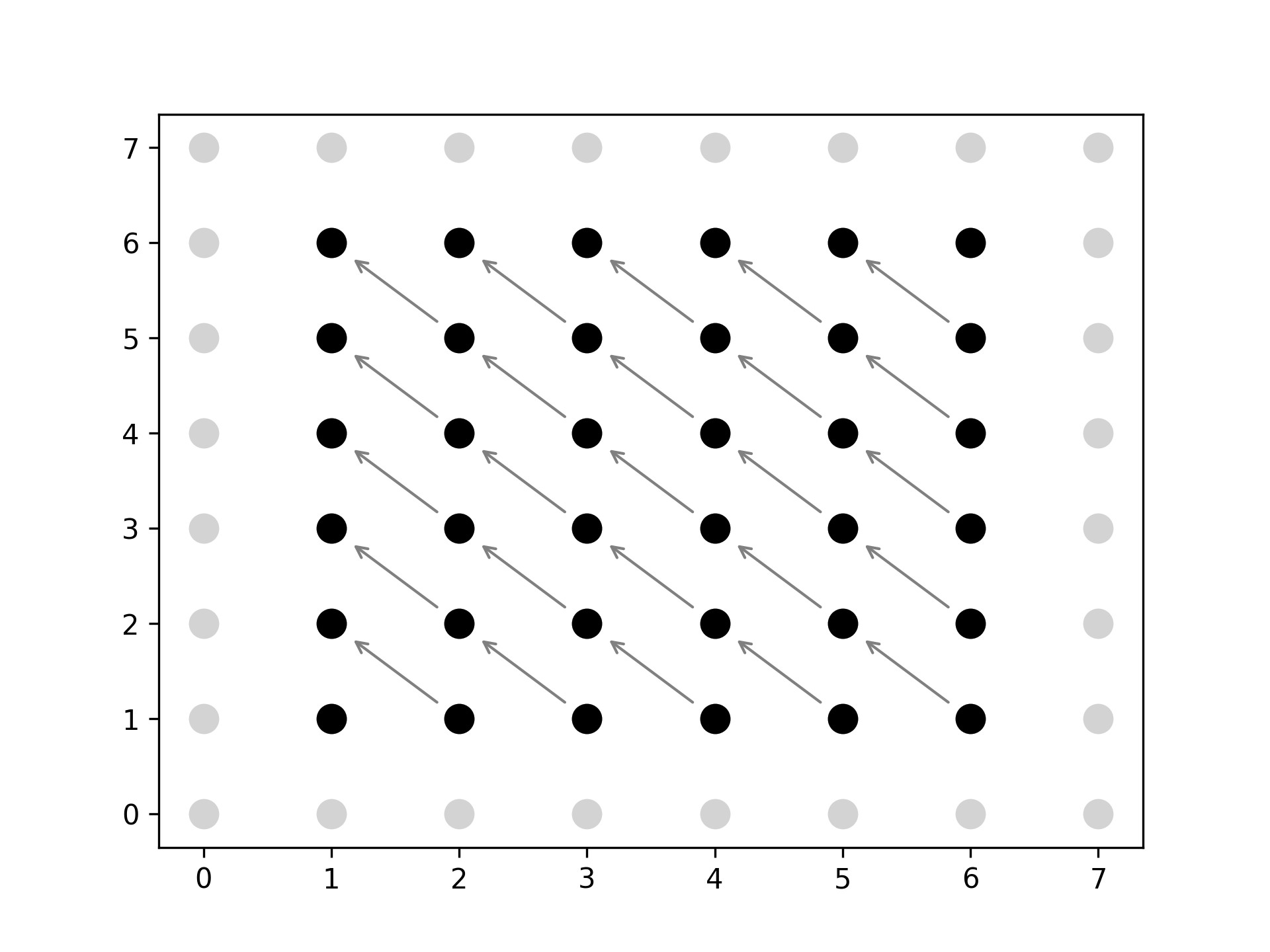
"Loop domain:"

[n] -> { S\_0[i, j] : 0 < i <= n and 0 < j <= n }

"R"

[n] -> { S\_0[i, j] -> S\_0[i' = i, j' = 1 + j] : 0 < i < n and 0 < j < n; S\_0[i, j] -> S\_0[i' = 1 + i, j' = j] : 0 < i <= -2 + n and 0 < j < n }

Rysunek grafu zależności 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 0 < j <= n }"

child:

schedule: "[n] -> [{ S\_0[i, j] -> [(i)] }, { S\_0[i, j] -> [(j)] }]"

permutable: 1

coincident: [ 1, 1 ]

W drzewie znajdują się dwa szeregowania:

* [n] -> {[i, j] -> [(i)] }, czyli transformacje afiniczne: c1=1, c2=0
* [n] -> {[i, j] -> [(j)] }, czyli transformacje afiniczne: c1=0, c2=1

Oznacza to, że istnieje możliwość implementacji techniki waveifronting.

**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 (wave-fronting) na poziomie iteracji.**

Aby utworzyć szeregowanie pozwalające na wave-fronting, należy zsumować prawe strony obydwu szeregowań. W ten sposób otrzymujemy: *i+j*.

Wywołana funkcja:

WAVE\_FR:=[n] -> { [i, j] -> [t=i+j] }\*LD;

daje poniższy wynik:

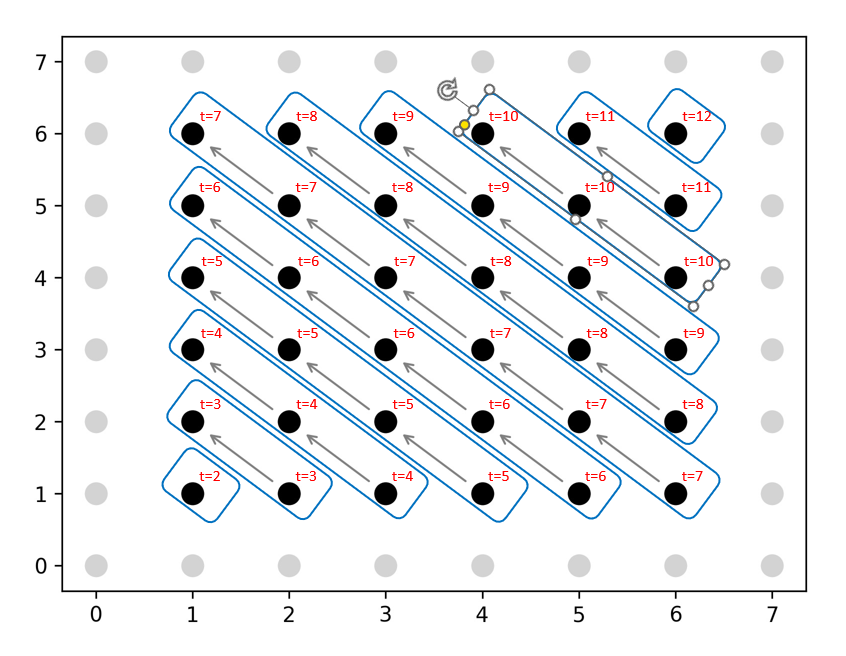
[n] -> { S\_0[i, j] -> [t = i + j] : 0 < i <= n and 0 < j <= n }

**Zadanie 5.**

**Stosując uzyskane w punkcie 4 szeregowanie za pomocą operatora scan znaleźć wszystkie partycje czasu dla przestrzeni 6x6 (12x12) i nanieść uzyskane partycje na rysunek utworzony w p.1.**

"scan (WAVE\_FR\*[n]->{:n=6})"

[n] -> { S\_0[i = 6, j = 6] -> [t = 12] : n = 6; S\_0[i = 5, j = 6] -> [t = 11] : n = 6; S\_0[i = 6, j = 5] -> [t = 11] : n = 6; S\_0[i = 4, j = 6] -> [t = 10] : n = 6; S\_0[i = 5, j = 5] -> [t = 10] : n = 6; S\_0[i = 6, j = 4] -> [t = 10] : n = 6; S\_0[i = 3, j = 6] -> [t = 9] : n = 6; S\_0[i = 4, j = 5] -> [t = 9] : n = 6; S\_0[i = 5, j = 4] -> [t = 9] : n = 6; S\_0[i = 6, j = 3] -> [t = 9] : n = 6; S\_0[i = 2, j = 6] -> [t = 8] : n = 6; S\_0[i = 3, j = 5] -> [t = 8] : n = 6; S\_0[i = 4, j = 4] -> [t = 8] : n = 6; S\_0[i = 5, j = 3] -> [t = 8] : n = 6; S\_0[i = 6, j = 2] -> [t = 8] : n = 6; S\_0[i = 1, j = 6] -> [t = 7] : n = 6; S\_0[i = 2, j = 5] -> [t = 7] : n = 6; S\_0[i = 3, j = 4] -> [t = 7] : n = 6; S\_0[i = 4, j = 3] -> [t = 7] : n = 6; S\_0[i = 5, j = 2] -> [t = 7] : n = 6; S\_0[i = 6, j = 1] -> [t = 7] : n = 6; S\_0[i = 1, j = 5] -> [t = 6] : n = 6; S\_0[i = 2, j = 4] -> [t = 6] : n = 6; S\_0[i = 3, j = 3] -> [t = 6] : n = 6; S\_0[i = 4, j = 2] -> [t = 6] : n = 6; S\_0[i = 5, j = 1] -> [t = 6] : n = 6; S\_0[i = 1, j = 4] -> [t = 5] : n = 6; S\_0[i = 2, j = 3] -> [t = 5] : n = 6; S\_0[i = 3, j = 2] -> [t = 5] : n = 6; S\_0[i = 4, j = 1] -> [t = 5] : n = 6; S\_0[i = 1, j = 3] -> [t = 4] : n = 6; S\_0[i = 2, j = 2] -> [t = 4] : n = 6; S\_0[i = 3, j = 1] -> [t = 4] : n = 6; S\_0[i = 1, j = 2] -> [t = 3] : n = 6; S\_0[i = 2, j = 1] -> [t = 3] : n = 6; S\_0[i = 1, j = 1] -> [t = 2] : n = 6 }



**Zadanie 6.**

**Wygenerować pseudokod i kod kompilowalny implementujący technikę fali frontowej.**

Wygenerowany pseudokod:

for (int c0 = 2; c0 <= 2 \* n; c0 += 1)

for (int c1 = max(1, -n + c0); c1 <= min(n, c0 - 1); c1 += 1)

(c1, c0 - c1);

Kod kompilowalny:

for (int c0 = 2; c0 <= 2 \* n; c0 += 1)

#pragma openmp parallel for

for (int c1 = max(1, -n + c0); c1 <= min(n, c0 - 1); c1 += 1)

a[c1][c0 - c1] = a[c1][c0 - c1-1] + a[c1 + 1][c0 - c1];

**Zadanie 7.**

**Zastosować program porównujący wyniki obliczeń (zadanie 7, L2) do sprawdzenia poprawności kodu docelowego w przestrzeni 6x6 (12x12).**

Wynik uruchomionej aplikacji porównującej:

Parallel code result:

00 01 02 03 04 05

00 01 03 06 10 15

00 01 03 06 10 15

00 01 03 06 10 15

00 01 03 06 10 15

00 01 03 06 10 15

Generated code result:

00 01 02 03 04 05

00 01 03 06 10 15

00 01 03 06 10 15

00 01 03 06 10 15

00 01 03 06 10 15

00 01 03 06 10 15

Results are identical.

**Zadanie 8.**

**Wygenerować kod reprezentujący kompilowalną pętlę całkowicie wymienną (patrz skrypt L4).**

Korzystając z wcześniej uzyskanego szeregowania:

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

Szeregowanie C pozwalające na wygenerowanie pętli całkowicie wymiennej:

C=(i,j)

Relacja:

FULL\_PERM:= [n]->{[I]->[C]};

gdzie I=(i,j), C=(i,j).

"codegen FULL\_PERM"

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

for (int c1 = 1; c1 <= n; c1 += 1)

S\_0(c0, c1);

**Zadanie 9.**

**Sprawdzić, że jest to pętla całkowicie wymienna poprzez obliczenie relacji zależności dla pętli wygenerowanej w p. 8 i zastosowanie operatora deltas do tej relacji. Wszystkie elementy uzyskanego wektora dystansu powinny być nieujemne.**

Relacja dla wygenerowanej pętli:

[n] -> { S\_0[c0, c1] -> S\_0[c0' = c0, c1' = 1 + c1] : 0 < c0 < n and 0 < c1 < n; S\_0[c0, c1] -> S\_0[c0' = 1 + c0, c1' = c1] : 0 < c0 <= -2 + n and 0 < c1 < n }

Loop domain:

[n] -> { S\_0[c0, c1] : 0 < c0 <= n and 0 < c1 <= n }

Deltas:

[n] -> { S\_0[c0 = 0, c1 = 1] : n >= 2; S\_0[c0 = 1, c1 = 0] : n >= 3 }

Wszystkie elementy uzyskanego wektora dystansu są nieujemne, co oznacza, że jest to pętla całkowicie wymienna.

**Załączniki:**

**Skrypt wykonujący 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 wavefronting

WAVE\_FR:=[n] -> { [i, j] -> [t=i+j] }\*LD;

print "WAVE\_FR";

WAVE\_FR;

#krok 5

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

scan (WAVE\_FR\*[n]->{:n=6});

##krok 6 wygenerowanie pseudokodu

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

print "CODE";

CODE;

scan (CODE\*[n]->{:n=2});

codegen CODE;

## krok 8

FULL\_PERM:= [n]-> {[i,j]->[i,j]}\*LD;

## generujemy pseudokod

print "codegen FULL\_PERM";

codegen FULL\_PERM;

**Kod aplikacji porównującej.**

// gcc -fopenmp 5-joined.c && ./a.out

#include <stdio.h>

#include <time.h>

#include <math.h>

#define ceild(n,d) ceil(((double)(n))/((double)(d)))

#define floord(n,d) floor(((double)(n))/((double)(d)))

#define max(x,y) ((x) > (y)? (x) : (y))

#define min(x,y) ((x) < (y)? (x) : (y))

int main() {

int n = 6;

int aInput[n+1][n+1];

int aGenerated[n+1][n+1];

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

for (int j = 0; j <= n; j++) {

aInput[i][j] = j;

aGenerated[i][j] = j;

}

// wejściowy

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

for (int j = 1; j <= n; j++) {

aInput[i][j] = aInput[i][j-1] + aInput[i+1][j];

}

}

// wygenerowany

for (int c0 = 2; c0 <= 2 \* n; c0 += 1)

#pragma openmp parallel for

for (int c1 = max(1, -n + c0); c1 <= min(n, c0 - 1); c1 += 1) {

aGenerated[c1][c0 - c1] = aGenerated[c1][c0 - c1-1] + aGenerated[c1+1][c0 - c1];

}

printf("Initial code result:\n");

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

printf("%02d ", aInput[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 (aInput[i][j] != aGenerated[i][j]) {

noMatchCount++;

// printf("Not matching at: [%d][%d]\n", i, j);

}

}

}

if (noMatchCount > 0)

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

else

printf("\nResults are identical.\n");

return 0;

}