Anna Plust, 36148
Automatyczna optymalizacja oprogramowania

Laboratorium 2: Znajdowanie zależności z wykorzystaniem PET i kalkulatora ISCC; generowanie kodu; implementacja postprocesora, opracowanie programu do porównania wyników produkowanych przez program źródłowy i docelowy.

Wariant pętli 4

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

Wariant szeregowania 4

```
SCHED:=[n]->{[i,j]->[i+3j][j]: 1<=i,j<=n}
```

Zadanie 1.
Dla wskazanej pętli za pomocą PET i kalkulatora ISCC znaleźć relację zależności, R.

```
"R"
[n] -> { S_3[i, j] -> S_1[i'] : 0 < i <= n and 0 < j <= n and i' > i and 0 < i' <= n;
S_3[i, j] -> S_3[i', j'] : 0 < i <= n and 0 < j <= n and i' > i and 0 < i' <= n and 0 <
j' <= n; S_3[i, j] -> S_3[i' = i, j'] : 0 < i <= n and 0 < j <= n and j' > j and 0 < j'
<= n; S_4[i] -> S_4[i'] : 0 < i <= n and i' > i and 0 < i' <= n; S_0[] -> S_4[i] : 0 < i
<= n; S_2[i, j] -> S_2[i' = -1 + 2i, j' = -1 + j] : i >= 2 and 2i <= n and 2 <= j <= n;
S_2[i = 1, j] -> S_2[i' = 1, j' = 1 + j] : 0 < j < n; S_1[i] -> S_3[i', j] : 0 < i <= n
and i' > i and 0 < i' <= n and 0 < j <= n; S_1[i] -> S_3[i' = i, j] : 0 < i <= n and 0 <
j <= n; S_1[i] -> S_1[i'] : 0 < i <= n and i' > i and 0 < i' <= n }
```

Zadanie 2.
Dla wskazanego szeregowania wygenerować pseudokod implementujący wave-fronting.

```
for (int c0 = 4; c0 <= 4 * n; c0 += 1)
  for (int c1 = max(1, floord(-n + c0 - 1, 3) + 1); c1 <= min(n, (c0 - 1) / 3); c1 += 1)
    (c0 - 3 * c1, c1);
```

Zadanie 3.
Opracować kod (na wejściu preprocesora), który przekształca pseudokod na kod kompilowalny.

```
#define P #pragma openmp parallel for
#define S a[i][j]=a[i][j-1];
#define i c0-c1
#define j c

int main() {
    int  c1,c2,n;
    int a[n*4-1][n];

    for (int c0 = 4; c0 <= 4 * n; c0 += 1) {
        P
        for (int c1 = max(1, floord(-n + c0 - 1, 3) + 1); c1 <= min(n, (c0 - 1) / 3); c1
+= 1) {
            S
```

```
            (c0 - 3 * c1, c1);
        }
    }
}
```

Zadanie 4.
Za pomocą preprocesora uzyskać kod docelowy (kompilowalny).
Po wykonaniu polecenia *gcc test.c -E -o test.txt* otrzymano następujący kod w pliku test.txt.

```
# 1 "test.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "test.c"

int main() {
    int c1,c2,n;
    int a[n*4-1][n];

    for (int c0 = 4; c0 <= 4 * n; c0 += 1) {
        #pragma openmp parallel for
        for (int c1 = max(1, floord(-n + c0 - 1, 3) + 1); c1 <= min(n, (c0 - 1) / 3); c1
+= 1) {
            a[c0-c1][c1]=a[c0-c1][c1-1];
            (c0 - 3 * c1, c1);
        }
    }
}
```

5. Opracować aplikację do porównania wyników produkowanych przez program źródłowy i docelowy.

app1.c

```
// gcc -fopenmp app1.c -lm && ./a.out

#include <stdio.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))

// #define P #pragma openmp parallel for
#define S   a[i][j]=a[i][j-1];
#define i c0-c1  // zgodnie z pierwszym elementem pseudoinstrtukcji (c0 - c1, c1);
#define j c1 // zgodnie z drugim elementem pseudoinstrtukcji (c0 - c1, c1);

int main() {
    int n = 6;
    int a[n*4][n];
```

```
    FILE *file = fopen("output1.txt", "w");
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    for (int c0 = 4; c0 <= 4 * n; c0 += 1) {
        #pragma openmp parallel for
        for (int c1 = max(1, floord(-n + c0 - 1, 3) + 1); c1 <= min(n, (c0 - 1) / 3); c1
+= 1) {
            S
            fprintf(file, "%d ", a[i][j]);
        }
        fprintf(file, "\n");
    }

    fclose(file);

    return 0;
}
```

app2.c

```
// gcc -fopenmp app2.c -lm && ./a.out

#include <stdio.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 a[n*4][n];

    FILE *file = fopen("output2.txt", "w");
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    for (int c0 = 4; c0 <= 4 * n; c0 += 1) {
        #pragma openmp parallel for
        for (int c1 = max(1, floord(-n + c0 - 1, 3) + 1); c1 <= min(n, (c0 - 1) / 3); c1
+= 1) {
            a[c0-c1][c1]=a[c0-c1][c1-1];
            fprintf(file, "%d ", a[c0-c1][c1]);
        }
        fprintf(file, "\n");
    }
```

```
    fclose(file);

    return 0;
}
```

comparator.c

```
// gcc comparator.c && ./a.out

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int compareFiles(const char *file1, const char *file2) {
    FILE *f1 = fopen(file1, "r");
    FILE *f2 = fopen(file2, "r");

    if (f1 == NULL || f2 == NULL) {
        perror("Error opening files");
        return -1;
    }

    int c1, c2;
    int position = 0;

    while ((c1 = fgetc(f1)) != EOF && (c2 = fgetc(f2)) != EOF) {
        position++;

        if (c1 != c2) {
            printf("Files differ at position %d\n", position);
            fclose(f1);
            fclose(f2);
            return position;
        }
    }

    if (c1 == EOF && c2 == EOF) {
        printf("Files are identical\n");
    } else {
        printf("Files have different lengths\n");
    }

    fclose(f1);
    fclose(f2);
    return 0;
}

int main() {
    const char *file1 = "output1.txt";
    const char *file2 = "output2.txt";

    int result = compareFiles(file1, file2);

    if (result == 0) {
```

```
        printf("Files are identical.\n");
    } else if (result == -1) {
        printf("Error opening files.\n");
    } else {
        printf("Files differ at position %d.\n", result);
    }

    return 0;
}
```