COL729 Major Exam Solutions
Compiler Optimization
Sem II, 2018-19

Answer all 5 questions Max. Marks: 50

1. Provide one example of a program and a data-flow analysis formulation where a region-based analysis would yield better solutions than data-flow analysis.
   a. Specify the data-flow analysis through the set of values, the partial-order operator, and the transfer functions
   b. Specify the composition, meet, and closure operators for the transfer functions for the region-based analysis
   c. Show an example, where the above-formulated region-based analysis would provide a more precise solution than the above-formulated data-flow analysis.

[10]

<u>Constant-propagation</u>

Data flow analysis

Domain : map (C) from program variable to a constant value
Bottom : NAC (not a constant)
Top : UI (uninitialized or not known)
Direction : Forward
Transfer function : For a statement (S), variable (x) and input map (in),
TF(S,in)(x) = identity, If S in not an assignment to x
else if S is an assignment to x, substitute the values present in input map (in) for each operand variable used by statement S,

- if any operand value is NAC => TF(S,in)(x) = NAC
- If any operand is UI => TF(S,in)(x) = UI
- else TF(S,in)(x) = value after substitution

Meet : If all predecessor send the variable to same constant C, then C else NAC

Region based analysis

Transfer function at exit of subregion S of a region R, $f_{R,OUT[s]}$ =
$\wedge$(Compose the transfer function for predecessor basic blocks B_i and S along all possible paths from entry of R to S)

Meet : $(f_1 \wedge_F f_2)(v) = f_1(v) \wedge f_2(v)$

Composition: $(f_1 \circ f_2)(v) = f_1(f_2(v))$

BB1->BB2
BB2->BB3
BB1->BB3

BB1:
x = 2;
y = 3;

BB2:
x = 3;
y = 2;

BB3:
z = x+y

- Using dataflow analysis at the OUT of BB3 will give z= NAC
- Using above described region based analysis at the OUT of BB3 will give z = 5

2. Consider the following loop nest:

```
for (ii = 0; ii < n; ii = ii+B)
    for (jj = 0; jj < n; jj = jj+B)
        for (kk = 0; kk < n; kk = kk+B)
            for (i = ii; i < ii+B; i++)
                for (j = jj; j < jj+B; j++)
                    for (k = kk; k < kk+B; k++)
                        Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
```

Express the iteration space and the data space of the loop-nest above through the four-tuples <**F**,**f**,**B**,**b**> to represent the data-spaces and the iteration space. How many matrices do you need to specify? What does each of them represent. What are the dimensions and the values of these matrices? [5]

Iteration space:

B,b for each loop nest specifying the lower and upper bound for iteration variable

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ -B & 0 & 0 & 1 & 0 & 0 & 0 \\ B & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -B & 0 & 0 & 1 & 0 & 0 \\ 0 & B & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -B & 0 & 0 & 1 & 0 \\ 0 & 0 & B & 0 & 0 & -1 & 0 \end{bmatrix} \quad i = \begin{bmatrix} ii` \\ jj` \\ kk` \\ i \\ j \\ k \\ n/B \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ -1 \\ 0 \\ -1 \\ 0 \\ -1 \\ 0 \\ B-1 \\ 0 \\ B-1 \\ 0 \\ B-1 \end{bmatrix}$$

Data space:

F,f for each static access specifying the affine function of loop index variables that produce the array index for all dimensions

Z[i , j]

$$F = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad i = \begin{bmatrix} \text{ii`} \\ \text{jj`} \\ \text{kk`} \\ i \\ j \\ k \\ n/B \end{bmatrix} \qquad f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

X[i , k]

$$F = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad i = \begin{bmatrix} \text{ii`} \\ \text{jj`} \\ \text{kk`} \\ i \\ j \\ k \\ n/B \end{bmatrix} \qquad f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Y[k , j]

$$F = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad i = \begin{bmatrix} \text{ii`} \\ \text{jj`} \\ \text{kk`} \\ i \\ j \\ k \\ n/B \end{bmatrix} \qquad f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

3. Consider the following loop-nest

```
for (i = 0; i <= 1000; i++)
  for (j = 0; j <= min(750,i); j++)
    X[j+1] = (1 / 3)  * (X[j] + X[j+1] + X[j+2]);
```

Use Fourier-Motzkin elimination to transform this loop with outer-axis k=i+j. Show the working of the algorithm succinctly, to show how you obtain the result. [8]

Iteration space:
$i \geq 0, i \leq 1000, j \geq 0, j \leq 750, j \leq i$

Introduce $k = i+j$

Let's eliminate variable j by using $j = k-i$

New iteration space constraints:
$i \geq 0, i \leq 1000, k-i \geq 0, k-i \leq 750, k-i \leq i$
or
$i \geq 0, i \leq 1000, k \geq i, k \leq i+750, k \leq 2*i$
or
$i \geq 0, i \leq 1000, i \leq k, i \geq k-750, 2*i \geq k$
(1) (2) (3) (4) (5)

k should be outer-axis, so project away i from constraints:

Using 1 and 2, $0 \leq 1000$
Using 1 and 3, $0 \leq k$
Using 4 and 2, $k-750 \leq 1000$ => $k \leq 1750$
Using 4 and 3, $k-750 \leq k$ => $-750 \leq 0$
Using 5 and 2, $k \leq 2*1000$ => $k \leq 2000$
Using 5 and 3, $k \leq 2*k$

Constraints for k:
$k \geq 0, k \leq 1750$

Constraints for i:
$i \geq 0, i \leq 1000, i \leq k, i \geq k-750, i \geq \lceil k/2 \rceil$

$i \geq \max(\lceil k/2 \rceil, k-750)$, $i \geq 0$ is redundant as $\lceil k/2 \rceil \geq 0$ for $k \geq 0$
$i \leq \min(k,1000)$

```
for (k = 0; k <= 1750; k++)
  for (i = max(⌈k/2⌉, k-750); i <= min(k,1000); i++)
    X[k-i+1] = (1 / 3)  * (X[k-i] + X[k-i+1] + X[k-i+2]);
```

Similarly can be done for eliminating i instead of j

4. Suppose there are two array accesses

A[2*i, j, i + j] and A[2*i + 4, j - 2, i + j]

In a 3-deep loop nest, with indices i, j, and k from the outer to the inner loop. What are all the types of temporal reuse (self-temporal and group-temporal) in this loop nest? Ignore spatial reuse.
[10]

Self-temporal reuse

For A[2*i, j, i+j],

$$F = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \qquad f = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

For A[2*i + 4, j-2, i+j],

$$F = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \qquad f = \begin{bmatrix} 4 \\ -2 \\ 0 \end{bmatrix}$$

In both the above cases,
Rank of F, r= 2 and loop nest depth d = 3
The difference between iteration variables (i,j,k), (i`,j`,k`) which access the same array index is given by nullspace of F :
2(i-i`) = 0, (j-j`) = 0; i.e. i = i`, j = j`
O(n) self temporal reuse, i.e. an element is accessed O(n) times where n is the num of iterations in loop with iteration variable k

Group-temporal reuse

The access have group temporal reuse, if there exists (i,j,k) and (i`,j`,k`), such that

$$F * ([i,j,k]^T - [i`,j`,k`]^T) = f_2 - f_1$$

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} i-i` \\ j-j` \\ k-k` \end{bmatrix} = \begin{bmatrix} 4 \\ -2 \\ 0 \end{bmatrix}$$

or

$i-i` = 2$

$j-j` = -2$

k-k` can be arbitrary from 0 to n, where n is the number of iterations in loop with iteration variable k

5. Consider the following program:

```
for (i = 0; i < n; i++) {
   A[i + 1] = A[i + 1]  * B[i + 1];  //S1
   for (j = i; j < n; j++) {
      C[i,j] = C[i,j] + D[0,i+1,2*j];  //S2
      A[j] = A[j] * C[i,j];  //S3
   }
}
```

Apply the algorithm to find parallelism with a constant number of synchronizations to this loop nest.

1. Show the queries you make to construct the program dependence graph. How many ILP queries did you have to make? [4]
2. After inserting a constant number of synchronizations, parallelize each separate loop nest (with no synchronization) if possible.
   a. What are the space-partition constraints? [3]
   b. Show the steps involved in solving the space-partition constraints. [3]
   c. What is the solution to your space-partition constraints [2]
   d. What is the generated code before eliminating empty iterations and tests from the inner loop (i.e., before applying Fourier Motzkin and before doing case-analysis)? [1]
   e. What is the generated code after applying Fourier Motzkin at each level of the iteration. Show the case analysis and the final generated code at each step. [4]

1. **ILP Queries (Total 9 queries, one for each pair of statements)**

   Edge between S1,S1: For both array A

   $\exists$ i,i\`, s.t. $0 \le i < n$, $0 \le i\` < n$, $i\` > i$, $i\` + 1 = i + 1$

   Edge between S2,S2: For both array C

   $\exists$ i,i\`,j,j\` s.t. $0 \le i < n$, $0 \le i\` < n$, $i \le j < n$, $i\` \le j\` < n$,
   $(i\`,j\`) > (i,j)$, $i\` = i$, $j\` = j$

   Edge between S3,S3: For both array A

   $\exists$ i,i\`,j,j\` s.t. $0 \le i < n$, $0 \le i\` < n$, $i \le j < n$, $i\` \le j\` < n$,
   $(i\`,j\`) > (i, j)$ $j\` = j$

   Edge between S1,S2 and S2,S1:
   No dependency as different arrays are accessed

Edge between S1,S3:
∃ i,i`,j, s.t. $0 \le i < n$, $0 \le i` < n$, $i` \le j < n$, $i < i`$, $i + 1 = j$

Edge between S3,S1:
∃ i,i`,j, s.t. $0 \le i < n$, $0 \le i` < n$, $i` \le j < n$, $i` < i$, $i + 1 = j$

Edge between S2,S3:
∃ i,i`,j,j` s.t. $0 \le i < n$, $0 \le i` < n$, $i \le j < n$, $i` \le j < n$, ((i < i`) or (i=i` and j ≤ j`))
i = i` and j = j`

Edge between S3,S2:
∃ i,i`,j,j` s.t. $0 \le i < n$, $0 \le i` < n$, $i \le j < n$, $i` \le j < n$, ((i` < i) or (i=i` and j` < j))
i = i` and j = j`

PDG:
S1 -> S3
S3 -> S1
S3 -> S3
S2 -> S3

2. **Constant number of synchronizations**:
We will have a separate loop nest for each SCC in the PDG.
In this case, S1,S3 form 1 SCC and S2 is another SCC. Further, S3 depends on S2, so S2 loop will be executed first and S3 will be executed later after synchronization

Loop1:
```
for (i = 0; i < n; i++) {
    for (j = i; j < n; j++) {
        C[i,j] = C[i,j] + D[0,i+1,2*j];  //S2
    }
}
```
barrier();
Loop2:
```
for (i = 0; i < n; i++) {
    A[i + 1] = A[i + 1]  * B[i + 1];  //S1
    for (j = i; j < n; j++) {
        A[j] = A[j] * C[i,j];  //S3
    }
}
```

Space partition constraints

For loop 1, S2 is not dependent on itself, so we can run all iterations in parallel.
Assuming a 2-D processor space,
The space partition constraints are:
For all (i,j) and (i`,j`) such that,
i,i` >= 0,   j >= i, j` >= i`,   i,i` < n,   j,j` < n, i = i`, j=j`

$$\begin{bmatrix} p1, & p2 \\ p3, & p4 \end{bmatrix} * \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} p5 \\ p6 \end{bmatrix} = \begin{bmatrix} p1', & p2' \\ p3', & p4' \end{bmatrix} * \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} p5' \\ p6' \end{bmatrix}$$

Substituting i = i` and j=j`, we get,
(p1-p1`) * i + (p2-p2`) * j + (p5-p5`) = 0
(p3-p3`) * i + (p4-p4`) * j + (p6-p6`) = 0

The simplest solution to this is

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} i \\ j \end{bmatrix}$$

or
$p_1 = i$ and $p_2 = j$

The processor space index variables $p_1 = i$ and $p_2 = j$ will have range
$0 \le p_1 < n$, and $i \le p_2 < n$,

Applying Fourier-motzkin to eliminate i from constraints for p_2, we get
$0 \le p_1 < n$, and $p_1 \le p_2 < n$,

For loop2, S3 is dependent on itself and S1 and S3 are also interdependent.

For S1 to S3 interdependence, the space-partition constraint imposed are:
For all (i) and (i`,j`) such that,

i,i` >= 0,   j` >= i`,   i,i` < n,   j` < n, i+1 = j`

$$\begin{bmatrix} p1 \\ p3 \end{bmatrix} * \begin{bmatrix} i \end{bmatrix} + \begin{bmatrix} p5 \\ p6 \end{bmatrix} = \begin{bmatrix} p1', & p2' \\ p3', & p4' \end{bmatrix} * \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} p5' \\ p6' \end{bmatrix}$$

p1 * i + p5 = p1` * i` + p2`*j` + p5`

p3 * i + p6 = p3` * i` + p4`*j` + p6`

or

p1 * i + p5 = p1` * i` + p2`*(i+1) + p5`
p3 * i + p6 = p3` * i` + p4`*(i+1) + p6`

or

i * (p1-p2`) + p5 -p2`-p5` = p1` * i`
i * (p3-p4`) + p6 -p6`-p4` = p3` * i`

or

p1 = p2`,  p3 = p4`, p5 = p2` + p5`, p6 = p6` + p4`, p1` = 0,  p3` = 0

The simplest solution for the above constraints are:

p1 = p2` = 1,  p3 = p4` = 1, p5 = 0, p5` = -1,  p6 = 0, p6` = -1, p1` = 0,  p3` = 0

For S2,

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} * [\, i \,] + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} i \\ i \end{bmatrix}$$

For S3,

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 0, & 1 \\ 0, & 1 \end{bmatrix} * \begin{bmatrix} i` \\ j` \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} j`-1 \\ j`-1 \end{bmatrix}$$

The processor space index variables for S1, $p_1 = p_2 = i$ will have range
$0 \le p_1,p_2 < n$

The processor space index variables for S3, $p_1 = p_2 = j-1$ will have range
$i-1 \le p_1,p_2 < n-1$

Applying Fourier-motzkin to eliminate i from constraints for p_1,p_2, we get
$-1 \le p_1,p_2 < n-1$

The generated code is:

<u>Loop1:</u>

```
for (p_1 = 0; p_1 < n; p_1++) {
   for (p_2 = p_1; p_2 < n; p_2++) {
      for (i = 0; i < n; i++) {
         for (j = i; j < n; j++) {
             if( p1 == i and p2 ==j)
                C[i,j] = C[i,j] + D[0,i+1,2*j];  //S2
} } } }
```

barrier();

<u>Loop2:</u>

```
for (p_1 = -1; p_1 < n; p_1++) {
   for (p_2 = -1; p_2 < n; p_2++) {
      for (i = 0; i < n; i++) {
         if(p_1 == i and p_1 == p_2)
            A[i + 1] = A[i + 1]  * B[i + 1];  //S1
         for (j = i; j < n; j++) {
            if(p_1 == j-1 and p_1 == p_2)
               A[j] = A[j] * C[i,j];  //S3
      }
   }
}
```

Fourier-Motzkin for S1

$p_1 \le i \le p_1$, $0 \le i < n$, $-1 \le p_1 < n$, $p_1 \le p_2 \le p_1$ $1 \le p_2 < n$

$\Rightarrow$ $0 \le p_1 < n$, $p_2 = p_1$

Fourier-Motzkin for S3

$0 \le i < n$, $p_1+1 \le j \le p_1+1$, $i \le j < n$, $-1 \le p_1 < n$, $p_1 \le p_2 \le p_1$ $1 \le p_2 < n$

$\Rightarrow$ $-1 \le p_1 < n-1$, $p_2 = p_1$

Loop1:

```
for (p1 = 0; p1 < n; p1++) {
   for (p2 = p1; p2 < n; p2++) {
      for (i = 0; i < n; i++) {
         for (j = i; j < n; j++) {
            if( p1 == i and p2 ==j)
               C[p1,p2] = C[p1,p2] + D[0, p1+1, 2*p2];  //S2
}}}}
```

barrier();

Loop2: *Case analysis*

```
p1 = -1;
for (i = 0; i < n; i++) {
   for (j = i; j < n; j++) {
      if(p1 == j-1)
        A[j] = A[j] * C[i, j ];  //S3
   }
}

for (p1 = 0; p1 < n-1; p1++) {
   for (i = 0; i < n; i++) {
      if(p1 == i)
         A[i + 1] = A[i + 1]  * B[i + 1];  //S1
      for (j = i; j < n; j++) {
         if(p1 == j-1)
            A[j] = A[j] * C[i,j];  //S3
   }
}

p1 = n-1
for (i = 0; i < n; i++) {
   if(p1 == i)
      A[i + 1] = A[i + 1]  * B[i + 1];  //S1
}
```

Fourier Motzkin

For S3 in case 1:

$i \le j \le n-1, \quad p_1+1 \le j \le p_1+1 \Rightarrow$

$i \le p_1+1, \; p_1+1 \le n-1, \; -1 \le p_1 \le -1 \Rightarrow$

$i \le 0, \; 1 \le n, \; , \; 0 \le i < n \Rightarrow i = 0, \; n > 0$

```
// p_1 = -1;
if(n > 0)  A[0] = A[0] * C[0, 0 ];  //S3
```

For S1 in case 2:

$0 \le i \le n-1, \quad p_1 \le i \le p_1 \Rightarrow$

$0 \le p_1, \; p_1 \le n-1, \; 0 \le p_1 \le n-2 \Rightarrow$

$0 \le n-1, \; 0 \le n-2 \Rightarrow n > 1$

For S3 in case 2:

$i \le j \le n-1, \quad p_1+1 \le j \le p_1+1 \Rightarrow$

$i \le p_1+1, \; 0 \le i \le n-1 \Rightarrow$

$0 \le i \le min(n-1, p_1+1) \Rightarrow$

For S1 in case 3:

$0 \le i \le n-1, \quad p_1 \le i \le p_1 \Rightarrow$

$0 \le p_1, \; p_1 \le n-1, \; n-1 \le p_1 \le n-1 \Rightarrow$

$0 \le n-1 \Rightarrow n > 1$

<u>Loop2:</u>

```
// p_1 = -1;
if(n > 0)  A[0] = A[0] * C[0, 0 ];  //S3

for (p_1 = 0; p_1 < n-1; p_1++) {
    A[p_1 + 1] = A[p_1 + 1]  * B[p_1 + 1];  //S1
    for (i = 0; i ≤ min(n-1, p_1 + 1); i++) {
        A[p_1 + 1] = A[p_1 + 1] * C[i,p_1 + 1];  //S3
    }
}

// p_1 = n-1
A[n] = A[n]  * B[n];  //S1
```