Exercise Sheet Solutions 11

Compiler Design Lab
Prof. Dr. Sebastian Hack
Fabian Ritter, B.Sc.

This is a solution proposal. It does not necessarily describe complete solutions but gives the initial ideas to solve the theoretical exercises and presents the results.

Exercise 11.1 Schedules

In the following, we use a slightly generalized notion of schedules compared to what we have seen in the lectures. Here, a schedule $\Theta : \Omega \to \mathbb{Z}^k$ is a function mapping statement instances to vectors of integers. A statement instance ω_1 is executed before another statement instance ω_2 according to a schedule Θ iff $\Theta(\omega_1) \prec_{lex} \Theta(\omega_2)$, where $\prec_{lex}$ is the lexicographic extension of the usual $<$ relation on integers.

1. Describe the difference between a statement and a statement instance and why this differentiation is useful.
2. Write down the schedule for a 2D loop nest that describes a loop interchange as a 2D matrix.
3. Write down the schedule for a 2D loop nest that describes a loop reversal of the inner loop as a 2D matrix.
4. How can the two schedules be combined? What does the combined scheduling matrix look like and what effect will it have on a 2D loop nest?
5. Describe when a schedule is considered legal with regards to a dependence relation Γ that contains all pairs of statement instances which depend on each other.

Solution:

1. A statement is a syntactic entity in the source program whereas a statement instance is a concrete evaluation of such a statement with given values for loop indices. Differentiating between statement instances and their corresponding statements is helpful as it allows to have more fine-grained dependence information which we can use for stronger optimizations.
2. The following linear function (which we can identify with its matrix) describes a loop interchange:
$$\Theta_1 : \begin{pmatrix} i \\ j \end{pmatrix} \mapsto \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix}$$
3. The following linear function (which we can identify with its matrix) describes a reversal of the innermost loop:
$$\Theta_2 : \begin{pmatrix} i \\ j \end{pmatrix} \mapsto \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix}$$
4. We can combine the transformations by composing their functions, i.e., by multiplying the corresponding matrices. We obtain two different combinations by either first applying loop reversal and then interchange:
$$\Theta_1 \cdot \Theta_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$
   ... or vice versa:
$$\Theta_2 \cdot \Theta_1 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$
5. A schedule Θ is respect with respect to Γ if $\forall \langle \omega_1 \leftarrow \omega_2 \rangle \in \Gamma.\ \Theta(\omega_1) \prec_{lex} \Theta(\omega_2)$ (i.e. each instance is executed after every instance that it depends on).

Exercise 11.2 Dependences

```
  for (int i = 0; i < N; i++)
    for (int j = 2; j < N; j++)
S:    C[j][i] = a * C[j-1][i] + b * C[j-2][i];
  for (int i = 0; i < N; i++)
P:  Out[i] = C[N-1][i];
```

1. Write down the iteration spaces (aka domain) of the statements S and P. Use a constraint system or visualize it in a 2D coordinate system.

2. Describe the dependences for the example above using dependence polyhedra. Characterize the dependences (RAW/WAR/WAW).

3. Describe a legal and beneficial transformation that could be applied and write down the transformed loop nest. Argue why the transformation is beneficial.

4. Which dimension in the example above (after your transformation) can be vectorized? What would the (pseudo) code look like?

Solution:

1. We can extract the following iteration domains from the source code:

$$S : 0 \leq i < N \wedge 2 \leq j < N$$

$$P : 0 \leq i < N$$

If we normalize this into a matrix, we get:

$$S : \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -2 \\ -1 & 0 & 1 & -1 \\ 0 & -1 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ N \\ 1 \end{pmatrix} \geq 0$$

$$P : \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} i \\ N \\ 1 \end{pmatrix} \geq 0$$

2. We can find loop-carried RAW dependences between instances of S and RAW dependences between instances of P and instances of S.

An instance $\binom{i}{j}$ of S depends on $\binom{i}{j-1}$ and $\binom{i}{j-2}$.

We obtain the following dependence polyhedra:

$$R_{S \leftarrow S'} : \{(i, j, i', j', N) \mid (j' = j + 1 \vee j' = j + 2) \wedge i = i' \wedge \binom{i}{j} \in D_S \wedge \binom{i'}{j'} \in D_S\}$$

and

$$R_{S \leftarrow P} : \{(i_S, j_S, i_P, N) \mid j_S = N - 1 \wedge i_P = i_S \wedge \binom{i_S}{j_S} \in D_S \wedge \binom{i_P}{} \in D_P\}$$

3. We can perform a loop interchange:

```
  for (int j = 2; j < N; j++)
    for (int i = 0; i < N; i++)
S:    C[j][i] = a * C[j-1][i] + b * C[j-2][i];
  for (int i = 0; i < N; i++)
P:  Out[i] = C[N-1][i];
```

This is beneficial since it causes subsequent instances of S to access subsequent memory cells which improves cache usage and prefetching. Furthermore, it enables vectorization.

4. We can vectorize both the S loop nest and the P loop nest in their i dimension:

```
  for (int j = 2; j < N; j++)
    for (int i = 0; i < N; i += w)
S:    C[j][i:i+w] = a *_w C[j-1][i:i+w] +_w b *_w C[j-2][i:i+w];
  for (int i = 0; i < N; i += w)
P:  Out[i:i+w] = C[N-1][i:i+w];
```

(assuming that N is a multiple of the vector width w.)