CS698L Assignment-2

Hritvik Taneja(160300)

September 9, 2019

Problem 1

Writes

1. $i_w, i_w - j_w$

Reads

1. $i_r, i_r - j_r - 1$
2. $i_r + 1, i_r - j_r$
3. $i_r - 1, i_r - j_r - 1$

Data Dependencies

$\mathbf{W_1}$ and $\mathbf{R_1}$

$$i_w, i_w - j_w = i_r, i_r - j_r - 1$$
$$i_w, j_w = i_r, j_r + 1$$

This is an anti dependence with $(0, 1)$ as the dependence vector.

$\mathbf{W_1}$ and $\mathbf{R_2}$

$$i_w, i_w - j_w = i_r + 1, i_r - j_r$$
$$i_w, j_w = i_r + 1, j_r + 1$$

This is an anti dependence with $(1, 1)$ as the dependence vector.

$\mathbf{W_1}$ and $\mathbf{R_3}$

$$i_w, i_w - j_w = i_r - 1, i_r - j_r - 1$$
$$i_w, j_w = i_r - 1, j_r$$

This is a flow dependence with $(1, 0)$ as the dependence vector.

Problem 2

Data dependencies and their permutations

$$\begin{bmatrix} t & i & j \\ 0 & 1 & -1 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} t & j & i \\ 0 & -1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} j & t & i \\ -1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} i & t & j \\ 1 & 0 & -1 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} i & j & t \\ 1 & -1 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} j & i & t \\ -1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix}$$

(a)

The above permutation matrices (i, j, t) and (t, j, i) imply that we cannot take the t^{th} and j^{th} to the innermost position, so the only valid loop unroll is possible for the j^{th} loop.

(b)

As we can see from the loop permutations above, we cannot permute any loop because it will make the direction matrix invalid.

(c)

Since none of the permutations are valid, not even for any 2 consecutive loops so no tiling is valid.

(d)

```
int i, j, t;
for (t = 0; t < 1024; t++) {
    for (j = 1; j < 2048; j++) {
        for (i = 0; i < min(1024, 2048-j); i++) {
            S(t, i, j);
        }
    }
}
```

(e)

```
int i, j, t;
for (t = 0; t < 1024; t++) {
    for (i = 0; i < 1024; i+=2) {
        for (j = 1; j < 2048 - i - 1; j++) {
            S(t, i, j);
            S(t, i+1, j);
        }
        S(t, i, 2048 - i - 1);
    }
}
```

Problem 3

Data dependencies

$$\begin{bmatrix} t & i & j & k \\ 1 & 0 & -1 & 1 \\ 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

(a)

- **Loop t**: It cannot be unrolled because taking this loop to the innermost position would make the direction matrix invalid $[(1, -1, 0, 1) \rightarrow (-1, 0, 1, 1)]$.
- **Loop i**: It cannot be unrolled because taking this loop to the innermost position would make the direction matrix invalid $[(0, 1, 0, -1) \rightarrow (0, 0, -1, 1)]$.
- **Loop j**: It can be unrolled because taking this loop to the innermost position would not make the direction matrix invalid.
- **Loop k**: It can be unrolled because taking this loop to the innermost position would not make the direction matrix invalid.

(b)

$(t, i, j, k) \rightarrow (t, i, k, j)$ and $(t, i, j, k) \rightarrow (t, j, i, k)$ are the only valid permutations and the reason for this are following.

- **Loop t** cannot be moved from the outermost position because it will make the direction matrix invalid for at-least on of the dependence.
- **Loop k** cannot be moved above **Loop i** because only the i^{th} loop has a positive direction for the 3^{rd} dependence.

(c)

From the previous part we know that $(t, i, j, k) \rightarrow (t, i, k, j)$ and $(t, i, j, k) \rightarrow (t, j, i, k)$ are the only valid permutations. Since all the permutations of **Loop j,k** and **Loop i,j** are valid so we can perform valid 2D tiling of the band ij and band jk.

(d)

Since **Loop t and i** are carrying all the dependencies, so we can parallelize **Loop j and k**.

(e)

```
int i, j, t, k;
for (t = 0; t < 1024; t++) {
    for (i = t; i < 1024; i++) {
        for (k = 1; k < i-1; k++) {
            for (j = max(t, k); j < i; j++) {
                S(t, i, j, k);
            }
        }
    }
}
```

Problem 4

(a)

A memory location $A[i][j]$ is accessed in the following order.

No.	Iteration	Read/Write
1	t-1, i-1, j	R
2	t-1, i, j-1	R
3	t-1, i, j	R
4	t-1, i, j	W
5	t-1, i, j+1	R
6	t-1, i+1, j	R
7	t, i-1, j	R
8	t, i, j-1	R
9	t, i, j	R
10	t, i, j	W
11	t, i, j+1	R
12	t, i+1, j	R

We will focus on the access between the two writes 4 and 10.

Output Dependence

Since a memory location is written at $(t-1, i, j)$ and (t, i, j), so there exists a flow dependence $(\mathbf{1}, \mathbf{0}, \mathbf{0})$.

Flow Dependence

Memory location written at 4 is read at $5, 6, 7, 8, 9$. All these access correspond to a flow dependence. Hence the flow dependencies for this program are $(\mathbf{0}, \mathbf{0}, \mathbf{1})$ $(\mathbf{0}, \mathbf{1}, \mathbf{0})$ $(\mathbf{1}, -\mathbf{1}, \mathbf{0})$ $(\mathbf{1}, \mathbf{0}, -\mathbf{1})$ $(\mathbf{1}, \mathbf{0}, \mathbf{0})$

Anti Dependence

Memory location written in 10 has already been read at $5, 6, 7, 8, 9$. All these access correspond to an anti dependence. Hence the anti dependencies for this program are $(\mathbf{1}, \mathbf{0}, -\mathbf{1})$ $(\mathbf{1}, -\mathbf{1}, \mathbf{0})$ $(\mathbf{0}, \mathbf{1}, \mathbf{0})$ $(\mathbf{0}, \mathbf{0}, \mathbf{1})$ $(\mathbf{0}, \mathbf{0}, \mathbf{0})$

(b)

Data dependencies and permutations

$$\begin{bmatrix} t & i & j \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} t & j & i \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} i & t & j \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} i & j & t \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} j & i & t \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} j & t & i \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix}$$

As we can see from all the permutations listed above only (t, i, j) and (t, j, i) are valid permutations.

(c)

The above permutation matrices (t, i, j) and (t, j, i) imply that we can take the i^{th} and j^{th} to the innermost position, so the only valid loop unroll are possible for the i^{th} and j^{th} loop.

(d)

From the previous part we know that $(t, i, j) \rightarrow (t, j, i)$ is a valid permutation. Since all the permutations of **Loop i,j** is valid so we can perform valid 2D tiling of the band ij.