HW1

Original program:

```
for j = 1 to n
    for i = 1 to n
        A[i, j] = A[i, j] + B[i-1, j] /* s1 */
        B[i, j] = A[i, j-1] * B[i, j] /* s2 */

```

(a)

Yes, the program is parallelizable by **Affine Space Partitions**.

(b)

Finding Space-Partition Constraints

Since there are 2 statements (s_1 and s_2), we have to find 2 affine partitions (one per statement).
Let

$$p = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + c_1$$
$$p = \begin{pmatrix} a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + c_2$$

be the one-dimensional affine partitions for the statements s_1 and s_2, respectively.

Note that the only data dependences in the code occur in:

1. Write access $A[i, j]$ in statement s_1 with read access $A[i, j-1]$ in statement s_2.
2. Write access $B[i, j]$ in statement s_2 with read access $B[i-1, j]$ in statement s_1.

The space-partition constraints imposed by the first dependence are:

For all (i, j) and (i', j') such that

$$\begin{array}{ll} 1 \le i \le n & 1 \le j \le n \\ 1 \le i' \le n & 1 \le j' \le n \\ i = i' & j = j' - 1 \end{array}$$

we have

$$p = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + c_1 = \begin{pmatrix} a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + c_2$$

Solving the equations, we have:

$$\begin{pmatrix} a_{11} - a_{21} & a_{12} - a_{22} \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + (c_1 - c_2 - a_{22}) = 0 \tag{$*$}$$

Since $(*)$ should hold for any (i, j), it must be that

$$\begin{aligned} a_{11} - a_{21} &= 0 \\ a_{12} - a_{22} &= 0 \\ c_1 - c_2 - a_{22} &= 0 \end{aligned}$$

Similarly, for the second dependence, we can also deduce the following constraints:

$$\begin{aligned} a_{11} - a_{21} &= 0 \\ a_{12} - a_{22} &= 0 \\ c_1 - c_2 + a_{21} &= 0 \end{aligned}$$

Simplifying all the constrains above together, we obtain:

$$a_{11} = a_{21} = -a_{22} = -a_{12} = c_2 - c_1$$

To allow parallelism, th coefficient matrix must have a nonzero rank. Thus we take, say, $a_{11} = a_{21} = 1, a_{22} = a_{12} = -1, c_2 = 0, c_1 = -1$.
According to this assignment, the (i, j)th iteration of s_1 is assigned to the processor $p = \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + (-1) = i - j - 1$, while the (i, j)th iteration of s_2 is assigned to the processor $p = \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + 0 = i - j$.

Finally, we can generate an equivalent code based on the affine partition above:

```
for p = -n to n-1
    for j = 1 to n
        for i = 1 to n
            if p == i-j-1
                A[i, j] = A[i, j] + B[i-1, j] /* s1 */
            if p == i-j
                B[i, j] = A[i, j-1] * B[i, j] /* s2 */
```

Eliminating Empty Iterations

The iteration space executed by partition p for statement s_1 is defined by

$$\begin{aligned} -n \le p &\le n-1 \\ 1 \le i &\le n \\ 1 \le j &\le n \\ j+p+1 &= i \end{aligned}$$

We can then tighten the bounds for each of the variables for statement s_1:

$$\begin{aligned} p: \quad & -n \le p \le n-2 \\ j: \quad & -p \le j \le n-p-1 \\ & 1 \le j \le n \\ i: \quad & i = j+p+1 \\ & 1 \le i \le n \end{aligned} \tag{1}$$

Similarly, we can also tighten the bounds for each of the variables for statement s_2:

$$\begin{aligned} p: \quad & -n+1 \le p \le n-1 \\ j: \quad & 1-p \le j \le n-p \\ & 1 \le j \le n \\ i: \quad & i = j+p \\ & 1 \le i \le n \end{aligned} \tag{2}$$

Given the tigher bounds, we can then rewrite the code such that it executes exactly in the union of iteration spaces:

```
for p = -n to n-1
    for j = max(1, -p) to min(n, n-p)
        for i = max(1, j+p) to min(n, j+p+1)
            if p == i-j-1
                A[i, j] = A[i, j] + B[i-1, j] /* s1 */
            if p == i-j
                B[i, j] = A[i, j-1] * B[i, j] /* s2 */
```

Eliminating Tests from Innermost Loops

Note that, by (1) and (2), s_1 and s_2 are mapped to the same set of processor ID's except for $p = -n$ and $p = n - 1$. Thus, we separate the partition space into 3 subspaces:

1. $p = -n$,
2. $-n + 1 \le p \le n - 2$, and
3. $p = n - 1$.

Subspace (1)

The loops degenerate into a single iteration, and thus the code for subspace (1) can reduced to:

```
/* p = -n
   j = n
   i = 1 */
A[1, n] = A[1, n] + B[0, n] /* s1 */
```

Subspace (3)

The loops degenerate into a single iteration, and thus the code for subspace (3) can reduced to:

```
/* p = n-1
   j = 1
   i = n */
B[n, 1] = A[n, 0] * B[n, 1] /* s2 */
```

Subspace (2)

Since $-n + 1 \le p \le n - 2$, we can first rewrite the code for subspace (2) as:

```
for p = -n+1 to n-2
    for j = max(1, -p) to min(n, n-p)
        for i = max(1, j+p) to min(n, j+p+1)
            if p == i-j-1
                A[i, j] = A[i, j] + B[i-1, j] /* s1 */
            if p == i-j
                B[i, j] = A[i, j-1] * B[i, j] /* s2 */
```

Notice that, by (1), statement s_1 is executed in the j''th iteration of loop with index j if and only if $\max(1, -p) \le j \le \min(n, n - p - 1)$. Also, we know by (2) that statement s_2 is executed in the j''th iteration of loop with index j if and only if
$\max(1, 1 - p) \le j \le \min(n, n - p)$.
As $n \ge 1$, it is clear that

$$\max(1, 1-p) - 1 \le \min(n, n-p-1).$$

Thus, we furthur split the loop with index j into 3 subspaces:

1. $\max(1, -p) \le j \le \min(\min(n, n-p-1), \max(1, 1-p) - 1)$, where only statement s_1 is executed,
2. $\max(\max(1, 1-p), \min(n, n-p-1) + 1) \le j \le \min(n, n-p)$, where only statement s_2 is executed, and
3. $\max(\max(1, -p), \max(1, 1-p)) \le j \le \min(\min(n, n-p-1), \min(n, n-p))$, where both statements s_1 and s_2 are executed.

Subspace (2-1): Only statement s_1 is executed

Therefore, for any $p \in [-n+1, n-2]$, ONLY statement s_1 (but not s_2) gets executed in the j'' th iteration of loop with index j iff

$$\begin{aligned}\max(1, -p) \le j'' &\le \min(\min(n, n-p-1), \max(1, 1-p) - 1) \\ &= \max(1, 1-p) - 1 \\ &= \max(0, -p),\end{aligned}$$

which holds iff

$$-p \le \max(1, -p) \le j'' \le -p.$$

This is equivalent to $p \le -1 \wedge j'' = -p$.

Subspace (2-2): Only statement s_2 is executed

On the other hand, for any $p \in [-n+1, n-2]$, ONLY statement s_2 (but not s_1) gets executed in the j'' th iteration of loop with index j iff

$$\begin{aligned}\min(n, n-p) \ge j'' &\ge \max(\max(1, 1-p), \min(n, n-p-1) + 1) \\ &= \min(n, n-p-1) + 1 \\ &= \min(n+1, n-p),\end{aligned}$$

which holds iff

$$n-p \ge \min(n, n-p) \ge j'' \ge n-p.$$

This is equivalent to $p \ge 0 \wedge j'' = n-p$.

Subspace (2-3): Both statements s_1 and s_2 are executed

Last but not least, for any $p \in [-n+1, n-2]$, both statements s_1 and s_2 get executed in the j'' th iteration of loop with index j iff

$$\max(\max(1, -p), \max(1, 1-p)) \le j'' \le \min(\min(n, n-p-1), \min(n, n-p))$$

, which can be rewritten as

$$\max(1, 1-p) \leq j'' \leq \min(n, n-p-1).$$

Hence, the code for subspace (2) can then be rewritten as:

```
for p = -n+1 to n-2
    /* subspace (2-1) */
    for j = max(1, -p) to max(1, 1-p)-1
        A[j+p+1, j] = A[j+p+1, j] + B[j+p, j] /* s1 */

    /* subspace (2-3) */
    for j = max(1, 1-p) to min(n, n-p-1)
        A[j+p+1, j] = A[j+p+1, j] + B[j+p, j] /* s1 */
        B[j+p, j] = A[j+p, j-1] * B[j+p, j] /* s2 */

    /* subspace (2-2) */
    for j = min(n, n-p-1)+1 to min(n, n-p)
        B[j+p, j] = A[j+p, j-1] * B[j+p, j] /* s2 */
```

By the conditions we obtain for subspace (2-1) and (2-2), most of the assignments to p in the 2 subspaces are evidently dead code and can be eliminated. To be precise,

- In subspace (2-1), statement s_1 is executed iff $p \leq -1 \wedge j = -p$.
- In subspace (2-2), statement s_2 is executed iff $p \geq 0 \wedge j = n - p$.

Therefore, we furthur eliminate the dead code and obtain the obtimized code for subspace (2):

```
for p = -n+1 to n-2
    /* subspace (2-1) */
    if p <= -1
        A[1, -p] = A[1, -p] + B[0, -p] /* s1 */

    /* subspace (2-3) */
    for j = max(1, 1-p) to min(n, n-p-1)
        A[j+p+1, j] = A[j+p+1, j] + B[j+p, j] /* s1 */
        B[j+p, j] = A[j+p, j-1] * B[j+p, j] /* s2 */

    /* subspace (2-2) */
    if p >= 0
        B[n, n-p] = A[n, n-p-1] * B[n, n-p] /* s2 */
```

Combining the code for all the subspaces, we obtain our final affine-partitioned code:

```
/* subspace (1) */
A[1, n] = A[1, n] + B[0, n] /* s1 */

/* subspace (2) */
for p = -n+1 to n-2
    /* subspace (2-1) */
    if p <= -1
        A[1, -p] = A[1, -p] + B[0, -p] /* s1 */

    /* subspace (2-3) */
    for j = max(1, 1-p) to min(n, n-p-1)
        A[j+p+1, j] = A[j+p+1, j] + B[j+p, j] /* s1 */
        B[j+p, j] = A[j+p, j-1] * B[j+p, j] /* s2 */

    /* subspace (2-2) */
    if p >= 0
        B[n, n-p] = A[n, n-p-1] * B[n, n-p] /* s2 */

/* subspace (3) */
B[n, 1] = A[n, 0] * B[n, 1] /* s2 */
```