Problem 1. Affine Transforms.

Consider the following program.

```c
for (int i = 1; i < n; i++) {
    for (int j = 0; j < n; j++) {
        A[i, j] = c * A[i, j-1];
    }
}

for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
    }
}
```

Assume \( A \) and \( B \) are two non-overlapping \( n \times n \) matrices. Both matrices are stored in row-major layout.

1. Draw the iteration space for the program. Use arrows to mark data-dependencies between iterations.

2. Parallelize each loop nest individually. Show the transformed code, and describe any transformation you performed.
Problem 1. Affine Transforms (cont.)

(Same program copied here for reference.)

```java
for (int i = 1; i < n; i++) {
    for (int j = 0; j < n; j++) {
        A[i, j] = c * A[i, j-1];
    }
}
for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
    }
}
```

3. Can you parallelize this program with no synchronization? Show the transformed code, or provide a justification that it is not possible.

**Approach 1**
(reasoning on data dependence graph)

```
\[ \text{L1: } p = c_i \text{i} + c_0 \]
\[ \text{L2: } p' = c_{ij'} + c_0' \]
```

**Approach 2:**
(affine partitioning)

```
\[ \text{L1: } A[i, j] \]
\[ \text{L2: } A[i', j'] \]
\[ A[i, j] \text{ & } A[i', j'] \text{ refer to the same location} \]
\[ \text{i.e. } i = i', \text{ } j = j'. \]
```

Then

\[ p(i, j) = p'(i', j') \]

\[ c_i \text{i} + c_0 = c_{ij'} + c_0' \]
\[ c_i - c_{ij'} + c_0 - c_0' = 0, \]
\[ c_i = 0, \text{ } c_{ij'} = 0, \text{ } c_0 = c_0'. \]

The fused loop: \( p = c_0 \)

\[ \implies \text{No parallelism.} \]
**Note on loop fusing:** (Supplementary, not covered in section)

1. Suppose there is no dependencies between Loop 1 & 2:
   - Loop 1: for (i) for (j)
     \[ A_{ij} = c \times A_{ij-1} \]
   - Loop 2: for (i) for (j)
     \[ B_{ij} = d \times B_{ij-1} \]
   
   Then \( p_1 = c, i + c_0 \)
   
   * Pick a solution for loop 1: \( p = i \)
   * Pick a solution for loop 2: \( p = j \)

   Loop fusion gives:
   
   for (p)
   
   for (i)
   
   for (j)
   
   if (p == i) \[ A_{ij} = \ldots \]
   
   for (i)
   
   for (j)
   
   if (p == j) \[ B_{ij} = \ldots \]

2. After applying optimizations to remove unnecessary loop variables:
   
   for (p)
   
   for (j) \[ A[p, j] = \ldots \]
   
   for (i) \[ B[i, p] = \ldots \]

   Loop fusion eliminates the synchronization barrier that existed between Loop 1 & 2.

2. Suppose loops are of different depth:
   
   Loop 1: for (i) for (j)
   
   Loop 2: for (i)
   
   \[ A_{ij} = c \times A_{ij} \]
   
\[ p = c_2 i + c_1 j + c_0, \quad p' = c_1 i' + c_0. \]

Dependencies: \[ \forall (i, j) = (i', 2). \]

\[ p(i, j) = p'(i'), \quad c_2 i + c_1 j + c_0 = c_1 i' + c_0. \]

\[ c_2 = c_1, \quad c_1 \cdot 2 + c_0 - c_0' = 0. \]

Two basis vectors: \[ [c_2, c_1] = [2, 1, 0], \quad [c_2, c_1] = [0, 2, 1]. \]

Two answers for \( p \): 1. \( p = i \) (loop 1), \( p = i' \) (loop 2)
2. \( p = j \) (loop 1), \( p = 2 \) (loop 2)

for (\( p_1 \))

for (\( p_2 \)) \( \{ \)

// degree-2 outermost parallelism.

for (\( i \))

for (\( j \))

if (\( p_1 = i \) \&\& \( p_2 = j \)) \[ A[i,j] = \ldots \]

for (\( i \))

if (\( p_1 = i \) \&\& \( p_2 = 2 \)) \[ B[i,j] = \ldots \]

\}

After optimization:

for (\( p_1 \))

for (\( p_2 \)) \( \{ \)

\[ A[p_1,p_2] = \ldots \]

if (\( p_2 = 2 \)) \[ B[i,j] = \ldots \]

\}
Problem 1. Affine Transforms (cont.)

(Same program copied here for reference.)

```java
for (int i = 1; i < n; i++) {
    for (int j = 0; j < n; j++) {
        A[i, j] = c * A[i, j-1];
    }
}
for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
    }
}
```

4. Can this program be written as a single loop nest with pipelined parallelism? Show the fully permutable loop nest and the best generated parallel code (using any necessary synchronization primitive, as in Homework 6), or provide a justification that it is not. You do not need to perform blocking.

- R. deep fully perm. ⇒ R-1 dag of 1-ism.

```java
for p=i: p<n; p++
    for j=0: j<n; j++
        if (p==1 or wait(t[p-1]≥j))
            A[i,j] = ...
            if (j≠0) B[i,j] = ...
            t[p]++;
```

- Sync variable t[p];
- proc. p has finished t[p] iterations.
Problem 1. Affine Transforms (cont.)

(Same program copied here for reference.)

```c
for (int i = 1; i < n; i++) {
    for (int j = 0; j < n; j++) {
        A[i, j] = c * A[i, j-1];
    }
}
for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
    }
}
```

5. Which of the original code, or any of the parallel versions, is likely to perform best? Assume any necessary blocking is now performed.

Paralleling individual loops is bad from a cache point of view.