DCC888 – Automatic Parallelization of Loops [1]

Name: _____ ID: _____

1. We have analyze only loops in which the induction variable is incremented by 1. This is not a serious limitation, because, in principle, most of the loops can be normalized in this way. For instance, consider the loop below:

   ```
   for (i = 2; i <= 100; i += 3)
       Z[i] = 0;
   ```

   This loop increments `i` by 3 each iteration; hence, setting to 0 each of `Z[2]`, `Z[5]`, `Z[8]`, ..., `Z[98]`. We can get the same effect with:

   ```
   for (j = 0; j <= 32; j++)
       Z[3*j + 2] = 0;
   ```

   In this exercise you must convert each of the loops below to a form where the induction variables are incremented by 1:

   (a)
   ```
   for (i = 10; i < 50; i = i + 7)
       X[i, i + 1] = 0;
   ```

   (b)
   ```
   for (i = -3; i <= 10; i = i + 2)
       X[i] = X[i + 1];
   ```

   (c)
   ```
   for (i = 50; i >= 10; i--)
       X[i] = 0;
   ```

[1] These exercises have been taken mostly from the earlier sections of Chapter 11 of the Dragon Book.

2. Compute the rank of each of the matrices below. Give both a maximal set of linearly independent columns and a maximal set of linearly independent rows.

$$\begin{bmatrix} 0 & 1 & 5 \\ 1 & 2 & 6 \\ 2 & 3 & 7 \\ 3 & 4 & 8 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 12 & 15 \\ 3 & 2 & 2 & 3 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 5 & 6 & 3 \end{bmatrix}$$

3. We write an n-dimensional array access inside k nested loops as the matrix product $\mathbf{E} \times \mathbf{I} + \mathbf{C}$, where $\mathbf{E}$ is an $n \times k$ matrix, and $\mathbf{I}$ and $\mathbf{C}$ are k element vectors. The matrix $\mathbf{I}$ is formed by the induction variables that control each of the k nested loops:

$$\begin{bmatrix} e_{11} & \dots & e_{1k} \\ \vdots & \ddots & \vdots \\ e_{n1} & \dots & e_{nk} \end{bmatrix} \times \begin{bmatrix} i_1 \\ \dots \\ i_k \end{bmatrix} + \begin{bmatrix} c_1 \\ \dots \\ c_k \end{bmatrix}$$

For instance, the access $a[i_1 + i_2, i_1 - 1]$ is written as:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} i_1 + i_2 \\ i_1 - 1 \end{bmatrix}$$

Below we have part of the code used in the LU decomposition of matrices. Write each array access in the loop below as an affine produce of matrices. One of the accesses has been done for you:

```
1  for (i = 0; i < N; ++i) {
2    B[i, i] = 1;
3    for (j = i + 1; j < N; ++j) {
4      B[j, i] = A[j, i] / A[i, i];
5      for (k = i + 1; k < N; ++k)
6        A[j, k] = A[i, k] - B[j, i] * A[i, k];
7    }
8    for (k = i; k < N; ++k)
9      C[i, k] = A[i, k];
10 }
```

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \times \begin{bmatrix} \text{i} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

4. Estimate the amount of data reuse in each access in the LU algorithm seen in the last exercise.

5. In each loop below, determine if there exists a dependence between any two array accesses. In case you find a dependence, explain which iteration k_0 reads data produced by which iteration k_1:

   (a)
   ```
   for (i = 1; i <= 10; i++)
       Z[i] = Z[i - 1];
   ```

   (b)
   ```
   for (i = 1; i < 10; i++)
       Z[2*i] = Z[2*i + 1];
   ```

   (c)
   ```
   for (i = 0; i <= 10; i++)
       for (j = 0; j <= 10; j++)
           Z[i, j] = Z[j + 10, i + 11];
   ```

6. Consider the loop below:

   ```
   for (i = 2; i < 100; i++)
       Z[i] = Z[i - 2];
   ```

   (a) What is the largest number of processors that can be used effectively to execute this loop?

   (b) Rewrite this loop in C for CUDA.