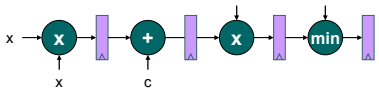
CprE / ComS 583 Reconfigurable Computing

Prof. Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University

Lecture #13 – Other Spatial Styles

Systolic Architectures

- Goal – general methodology for mapping computations into hardware (spatial computing) structures
- Composition:
  - Simple compute cells (e.g. add, sub, max, min)
  - Regular interconnect pattern
  - Pipelined communication between cells
  - I/O at boundaries

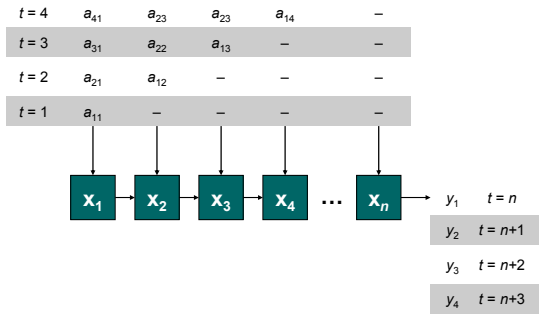


Example – Matrix-Vector Product

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

```
for (i=1; i<=n; i++)
  for (j=1; j<=n; j++)
    y[i] += a[i][j] * x[j];
```


Matrix-Vector Product (cont.)



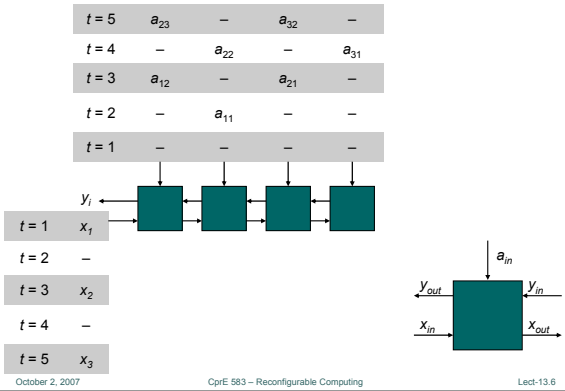
Banded Matrix-Vector Product

$$\begin{bmatrix} a_{11} & a_{12} & & & 0 \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & & \ddots & \\ 0 & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_n \end{bmatrix}$$

(q = 2, p = 3)

```
for (i=1; i<=n; i++)
  for (j=1; j<=p+q-1; j++)
    y[i] += a[i][j-q-i] * x[j];
```


Banded Matrix-Vector Product (cont.)



Outline

- Recap
- Non-Numeric Systolic Examples
- Systolic Loop Transformations
  - Data dependencies
  - Iteration spaces
  - Example transformations
- Reading – Cellular Automata
- Reading – Bit-Serial Architectures

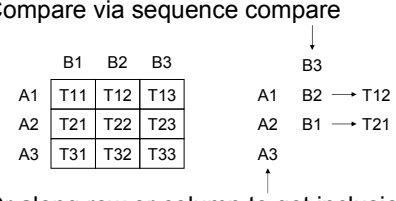
Example – Relational Database

- Relation is a collection of tuples that all have the same attributes
  - Tuple is a fixed number of objects
  - Represented in a table

tuple #	Name	School	Age	QB Rating
0	T. Brady	Michigan	30	141.8
1	T. Romo	E. Illinois	27	112.9
2	J. Delhomme	LA-Lafayette	32	111.9
3	P. Manning	Tennessee	31	110.4
4	R. Grossman	Florida	27	45.2

Database Operations

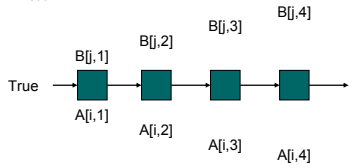
- Intersection: $A \cap B$ – all records in both relation A and B
- Must compare all $|A|$ x $|B|$ tuples
- Compare via sequence compare



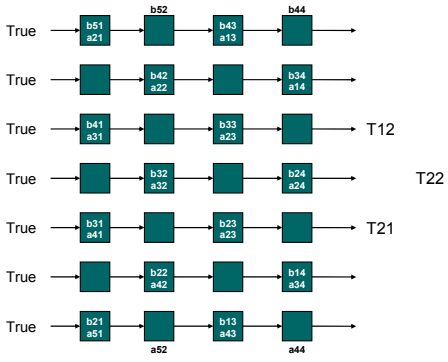
- Or along row or column to get inclusion bitvector

Database Operations (cont.)

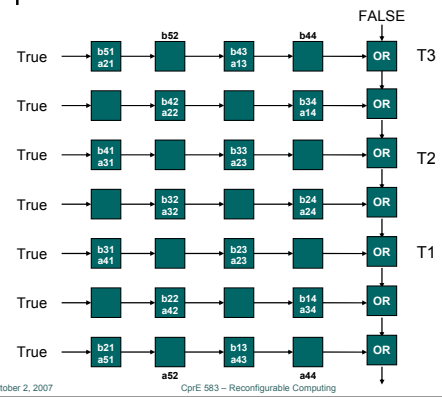
- Tuple Comparison
  - Problem – tuples are long, comparison time might limit computation rate
  - Strategy – perform comparison in pipelined manner by fields
    - Stagger fields
    - Arrange to compute field i on cycle after i-1
  - Cell: $t_{out} = t_{in}$ and a_{in} xnor b_{in}



Database Intersection



Database Intersection (cont.)



Database Operations (cont.)

- Unique: remove duplicate elements in multirelation *A*
  - Intersect *A* with *A*
- Union: *A* U *B* – one copy of all tuples in *A* and *B*
  - Concatenate *A* and *B*
  - Use Unique to remove duplicates
- Projection: collapse *A* by removing select fields of every tuple
  - Sample fields in *A*'
  - Use Unique to remove duplicates

Database Join

- Join: $AJ_{C_A,C_B}B$ – where columns C_A in A intersect columns C_B in B, concatenate tuple A_i and B_j
  - Match C_A of A with C_B of B
  - Keep all $T_{i,j}$
  - Filter i,j for which $T_{i,j}$ = true
  - Construct join from matched pairs
- Claim: Typically, $| AJ_{C_A,C_B}B | << | A | | B |$

Database Summary

- Input database – $O(n)$ data
- Operations require $O(n^2)$ data
  - $O(n)$ if sorted first
  - $O(n \log(n))$ to sort
- Systolic implementation – works on $O(n)$ processing elements in $O(n)$ time
- Typical database [KunLoh80A]:
  - 1500 bit tuples
  - 10,000 records in a relation
  - ~1 4-LUT per bit-compare
    - ~1600 XC4062 FPGAs
    - ~84 XC4LX200 FPGAs

Systolic Loop Transformations

- Automatically re-structure code for
  - Parallelism
  - Locality
- Driven by dependency analysis

Defining Dependencies

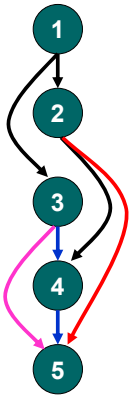
• Flow Dependence	W → R	δ^f	true
• Anti-Dependence	R → W	δ^a	false
• Output Dependence	W → W	δ^o	false
• Input Dependence	R → R	δ^i	

S1) a = 0;
S2) b = a;
S3) c = a + d + e;
S4) d = b;
S5) b = 5+e

Example Dependencies

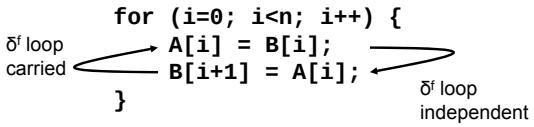
S1) a = 0;
S2) b = a;
S3) c = a + d + e;
S4) d = b;
S5) b = 5+e

S1 δ^f S2	**due to a**
S1 δ^f S3	**due to a**
S2 δ^f S4	**due to b**
S3 δ^a S4	**due to d**
S4 δ^a S5	**due to b**
S2 δ^o S5	**due to b**
S3 δ^i S5	**due to e**

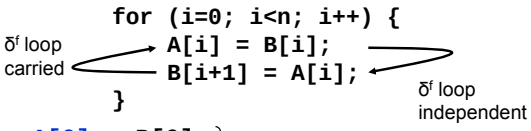


Data Dependencies in Loops

- Dependence can flow across iterations of the loop
- Dependence information is annotated with iteration information
- If dependence is across iterations it is loop carried otherwise loop independent



Unroll Loop to Find Dependencies




```
A[0] = B[0];
B[1] = A[0];      i = 0
A[1] = B[1];
B[2] = A[1];      i = 1
A[2] = B[2];
B[3] = A[2];      i = 2
  ...
```

Distance/direction of dependence is also important

Thought Exercise

- Consider the Laplace Transformation: $L(f) = F(s) = \int_0^{\infty} s^{-st} f(t)\, dt$

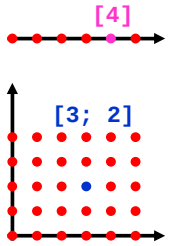
```
for (i=1; i<N; i++)
  for (j=1; j<N; j++)
    c = -4*a[i][j] + a[i-1][j] + a[i+1][j];
    c += a[i][j+1] + a[i][j+1]
    b[i][j] = c;
  }
}
```

- In teams of two, try to determine the flow dependencies, anti dependencies, output dependencies, and input dependencies
  - Use loop unrolling to find dependencies
- Most dependencies found gets a prize

Iteration Space

- Every iteration generates a point in an *n*-dimensional space, where *n* is the depth of the loop nest

```
for (i=0; i<n; i++) {
    ...
}
for (i=0; i<n; i++) {
    for (j=0; j<5; j++) {
      ...
    }
}
```



Distance Vectors

```
for (i=0; i<n; i++) {
    A[i] = B[i];
    B[i+1] = A[i];
}
```

```
A[0] = B[0];
B[1] = A[0];      i = 0
A[1] = B[1];
B[2] = A[1];      i = 1
A[2] = B[2];
B[3] = A[2];      i = 2
  ...
```

Distance vector is the difference between the target and source iterations

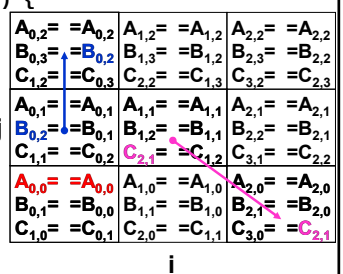
$$d = I_t - I_s$$

Exactly the distance of the dependence, i.e.,

$$I_s + d = I_t$$

Distance Vectors Example

```
for (i=0; i<n; i++) {
  for (j=0; j<m; j++) {
    A[i,j] =   ;
      = A[i,j];
    B[i,j+1] =  ;
      = B[i,j];
    C[i+1,j] =  ;
      = C[i,j+1];
}
```



A yields [0; 0]

B yields [0; 1]

C yields [1; -1]

FIR Distance Vectors

```
for (i=0; i<n; i++)
  for (j=0; j<m; j++)
    Y[i] = Y[i]+X[i+j]*W[j];
```

Y yields: δ^a [0; 0]

Y yields: δ^f [0; 1]

X yields: δ^i [1; -1]

W yields: δ^i [1; 0]

$Y_0=$ $=Y_0$ $=X_3$ $=W_3$	$Y_1=$ $=Y_1$ $=X_4$ $=W_3$	$Y_2=$ $=Y_2$ $=X_5$ $=W_3$	$Y_3=$ $=Y_3$ $=X_6$ $=W_3$
$Y_0=$ $=Y_0$ $=X_2$ $=W_2$	$Y_1=$ $=Y_1$ $=X_3$ $=W_2$	$Y_2=$ $=Y_2$ $=X_4$ $=W_2$	$Y_3=$ $=Y_3$ $=X_5$ $=W_2$
$Y_0=$ $=Y_0$ $=X_1$ $=W_1$	$Y_1=$ $=Y_1$ $=X_2$ $=W_1$	$Y_2=$ $=Y_2$ $=X_3$ $=W_1$	$Y_3=$ $=Y_3$ $=X_4$ $=W_1$
$Y_0=$ $=Y_0$ $=X_0$ $=W_0$	$Y_1=$ $=Y_1$ $=X_1$ $=W_0$	$Y_2=$ $=Y_2$ $=X_2$ $=W_0$	$Y_3=$ $=Y_3$ $=X_3$ $=W_0$

Re-label / Pipeline Variables

- Remove anti-dependencies and input dependencies by relabeling or pipelining variables

```
for (i=0; i<n; i++) {
  for (j=0; j<m; j++) {
    W^i[j] = W^i-1[j];
    X^i[i+j]=X^i-1[i+j];
    Y^j[i] = Y^j-1[i]+X^i[i+j]*W^i[j];
  }
}
```

$$D = \begin{matrix} Y & W & X \end{matrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

- Creates new flow dependencies
- Removes anti/input dependencies

FIR Dependencies

$$D = \begin{matrix} Y & W & X \end{matrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

$Y^2_0=$ $=Y^1_0$ $X^0_2=$ $=X^{-1}_2$ $W^0_2=$ $=W^{-1}_2$	$Y^2_1=$ $=Y^1_1$ $X^1_3=$ $=X^0_3$ $W^1_2=$ $=W^0_2$	$Y^2_2=$ $=Y^1_2$ $X^2_4=$ $=X^1_4$ $W^2_2=$ $=W^1_2$
$Y^1_0=$ $=Y^0_0$ $X^0_1=$ $=X^{-1}_1$ $W^0_1=$ $=W^{-1}_1$	$Y^1_1=$ $=Y^0_1$ $X^1_2=$ $=X^0_2$ $W^1_1=$ $=W^0_1$	$Y^1_2=$ $=Y^0_2$ $X^2_3=$ $=X^1_3$ $W^2_1=$ $=W^1_1$
$Y^0_0=$ $=Y^{-1}_0$ $X^0_0=$ $=X^{-1}_0$ $W^0_0=$ $=W^{-1}_0$	$Y^0_1=$ $=Y^{-1}_1$ $X^1_1=$ $=X^0_1$ $W^0_2=$ $=W^0_0$	$Y^0_2=$ $=Y^{-1}_2$ $X^2_2=$ $=X^1_2$ $W^2_0=$ $=W^1_0$

j

i

Transforming to Time and Space

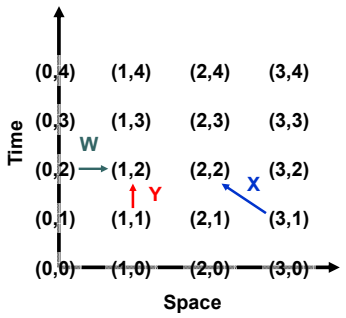
- Using data dependencies, find T
- T defines a mapping of the iteration space into a time component π, and a space component, S
- T = [π; S]
  - If $\pi \cdot l_1 = \pi \cdot l_2$, then l_1 and l_2 execute at the same time
  - $\pi \cdot d$ – amount of time units to move data items ($\pi \cdot d > 0$)
  - Any S can be picked that makes T a bijection
- See [Mol83A] for more details

Calculating T for FIR

- For $\pi = [p_1\ p_2]$
- Since $\pi \cdot d > 0$, we see that:
  - p_2 != 0 (from Y)
  - p_1 != 0 (from W)
  - $p_1 > p_2$ (from X)
- Smallest solution π = [2 1]
- S can be [1 0], [0 1], [1 1]

$$D = \begin{matrix} Y & W & X \end{matrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

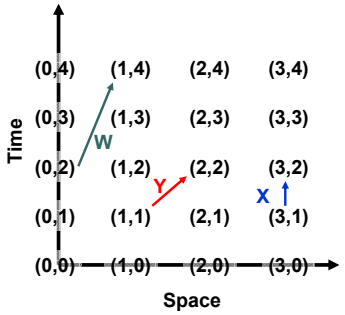
An Example Transformation



$$D = \begin{matrix} Y & W & X \end{matrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

An Example Transformation (cont.)

$$T = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$



$$T \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 2i + j \\ i + j \end{bmatrix}$$

$$T \cdot D = \begin{matrix} \text{Y W X} \\ \begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

Summary

- Non-numeric (database ops) example of systolic computing
  - Multiple use of each input data item
  - Concurrency
  - Regular data and control flow
- Loop transformations
  - Data dependency analysis
  - Restructure code for parallelism, locality