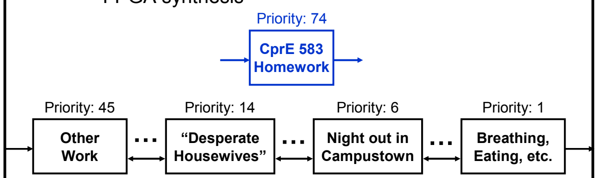
CprE / ComS 583
Reconfigurable Computing

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

Lecture #12 – Other Spatial Styles

Quick Points

- HW #3 coming out today
  - Due Tuesday, October 17 (midnight)
    - Systolic computing structures
    - Systolic mapping
    - Logic partitioning
    - FPGA synthesis



Project Proposals

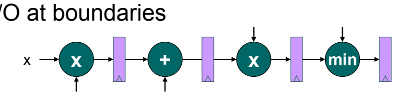
- Due Sunday, 10/8 at midnight
  - Purpose – to provide a background and overview of the project
  - Goal – allow me to understand what you are intending to do
- Project topic:
  - Perform an in-depth exploration of some area of reconfigurable computing
  - Whatever topic you choose, you **must** include a strong experimental element in your project
  - Work in groups of 2+ (3 if very lofty proposal)

Project Proposals (cont.)

- Suggested structure [3-4 pages, IEEE conf. format]
  - **Introduction** – what is the context for this work? What problem are you trying to address? Why is it interesting/challenging?
  - **Prior work** – what is the related work? How does your work differ from these? (5-10 references)
  - **Approach** – how are you going to tackle the problem? What tools and methodologies do you intend on using? What experiments do you intend on running?
  - **Expected results** – what do you expect the outcome of your project to be? What are the deliverables? How do you intend on presenting your results?
  - **Milestones** – what is your expected progress schedule? Provide a weekly / bi-weekly basis

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 – Finite Impulse Response

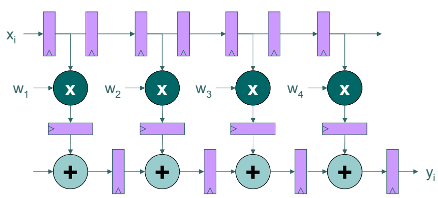
- A Finite Impulse Response (FIR) filter is a type of digital filter
  - Finite – response to an impulse eventually settles to zero
  - Requires no feedback

$$y_i = w_1 \cdot x_i + w_2 \cdot x_{i+1} + \cdots + w_k \cdot x_{i+k-1}$$
$$= \sum_{j=1}^{k} w_i \cdot x_{i+j-1}$$

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

Finite Impulse Response (cont.)

- Sequential
  - Memory bandwidth per output – $2k+1$
  - $O(k)$ cycles per output
  - $O(1)$ hardware
- Systolic
  - Memory bandwidth per output – 2
  - $O(1)$ cycles per output
  - $O(k)$ hardware

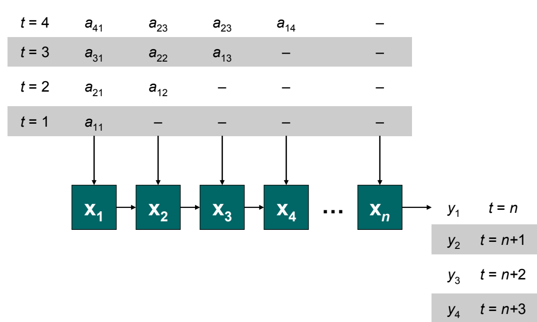


Example – Matrix-Vector Product

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

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


Matrix-Vector Product (cont.)



Outline

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

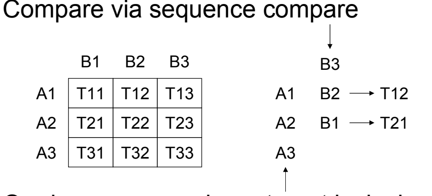
Example – Relational Database

- 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	D. Carr	Fresno State	27	113.6
1	P. Rivers	NC State	24	107.4
2	D. McNabb	Syracuse	29	105.3
3	C. Pennington	Marshall	30	103.4
4	R. Grossman	Florida	26	100.9

Database Operations

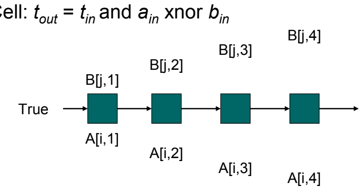
- Intersection: $A \cap B$ – all records in both relation A and B
- Must compare all $|A|$ x $|B|$ tuples
- Compare via sequence compare



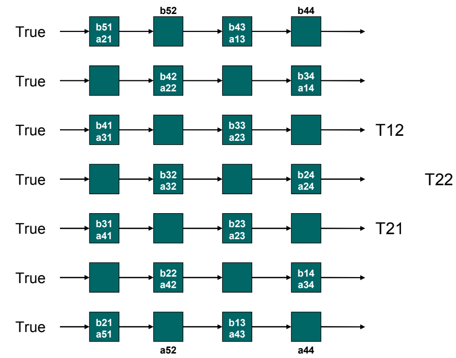
- Or along row or column to get inclusion bitvector

Database Operations (cont.)

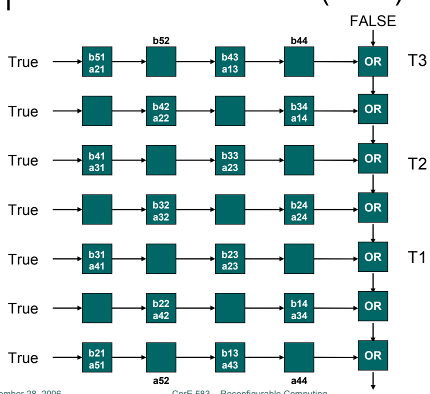
- Tuple Comparison
  - Problem – tuples are long, comparison time might limit computation rate
  - Strategy – perform comparison in pipelined manner by fields
    - Stagger fields
    - Arrange to compute field *i* on cycle after *i*-1
  - Cell: $t_{out} = t_{in}$ and a_{in} xnor b_{in}



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

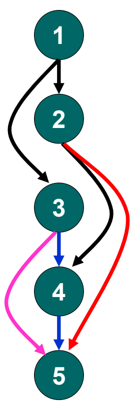
Flow Dependence	W → R	δ^f	true
Anti-Dependence	R → W	δ^a	false
Output Dependence	W → W	δ^o	false
Input Dependence	R → R	δ^i	

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

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

δ^f loop carried

δ^f loop independent

Unroll Loop to Find Dependencies

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

δ^f loop carried

δ^f loop independent

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

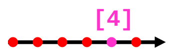
```
for (i=1; i<N; i++)
  for (j=1; j<N; j++)
    c = -4*a[i][j] + a[i-1][j] + a[i+1][j];
    c += a[i][j+1] + a[i][j+1]
    b[i][j] = c;
  }
}
```

- 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

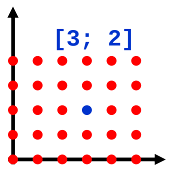
Iteration Space

- 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];
}
```

$\left.\begin{array}{l}A[0] = B[0];\\ B[1] = A[0];\end{array}\right\}$ i = 0

$\left.\begin{array}{l}A[1] = B[1];\\ B[2] = A[1];\end{array}\right\}$ i = 1

$\left.\begin{array}{l}A[2] = B[2];\\ B[3] = A[2];\end{array}\right\}$ i = 2

...

Distance vector is the difference between the target and source iterations

$$\mathbf{d} = \mathbf{I}_t - \mathbf{I}_s$$

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

$$\mathbf{I}_s + \mathbf{d} = \mathbf{I}_t$$

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];
}
```

j			
	$A_{0,2}$= =$A_{0,2}$ $B_{0,3}$= =$B_{0,2}$ $C_{1,2}$= =$C_{0,3}$	$A_{1,2}$= =$A_{1,2}$ $B_{1,3}$= =$B_{1,2}$ $C_{2,2}$= =$C_{1,3}$	$A_{2,2}$= =$A_{2,2}$ $B_{2,3}$= =$B_{2,2}$ $C_{3,2}$= =$C_{2,3}$
	$A_{0,1}$= =$A_{0,1}$ $B_{0,2}$= =$B_{0,1}$ $C_{1,1}$= =$C_{0,2}$	$A_{1,1}$= =$A_{1,1}$ $B_{1,2}$= =$B_{1,1}$ $C_{2,1}$= =$C_{1,2}$	$A_{2,1}$= =$A_{2,1}$ $B_{2,2}$= =$B_{2,1}$ $C_{3,1}$= =$C_{2,2}$
	$A_{0,0}$= =$A_{0,0}$ $B_{0,1}$= =$B_{0,0}$ $C_{1,0}$= =$C_{0,1}$	$A_{1,0}$= =$A_{1,0}$ $B_{1,1}$= =$B_{1,0}$ $C_{2,0}$= =$C_{1,1}$	$A_{2,0}$= =$A_{2,0}$ $B_{2,1}$= =$B_{2,0}$ $C_{3,0}$= =$C_{2,1}$
		i	

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];
  }
}
```

$$D = \begin{array}{c} \begin{array}{ccc} \mathbf{Y} & \mathbf{W} & \mathbf{X} \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \end{array}$$

- Creates new flow dependencies
- Removes anti/input dependencies

FIR Dependencies

$$D = \begin{array}{c} \begin{array}{ccc} \mathbf{Y} & \mathbf{W} & \mathbf{X} \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \end{array}$$

j			
	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
		i	

Transforming to Time and Space

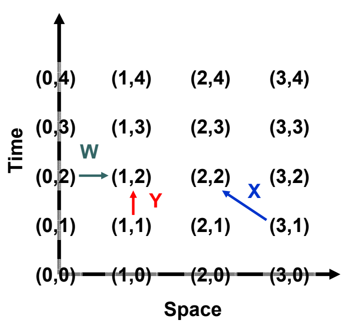
- Using data dependencies, find T
- T defines a mapping of the iteration space into a time component π, and a space component, S
- T = [π; S]
  - If $\pi \cdot I_1 = \pi \cdot I_2$, then I_1 and I_2 execute at the same time
  - $\pi \cdot d$ – amount of time units to move data items ($\pi \cdot d > 0$)
  - 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]

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

An Example Transformation

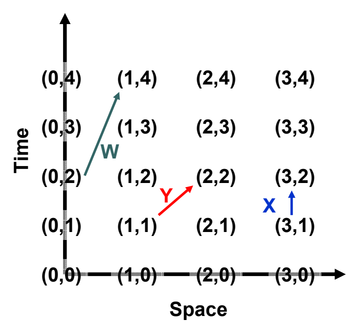


$$D = \begin{array}{c} \begin{matrix} Y & W & X \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \end{array}$$

An Example Transformation (cont.)

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

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



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

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