



# Concurrent Models of Computation for Embedded Software

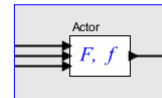
Edward A. Lee

Professor, UC Berkeley  
EECS 290n – Advanced Topics in Systems Theory  
Fall, 2004

Copyright © 2004, Edward A. Lee, All rights reserved

Lecture 16: Statically Schedulable Dataflow

## Execution Policy for a Dataflow Actor



Suppose  $s \in S^n$  is a concatenation of firing rules,

$$s = u_1 \cdot u_2 \cdot u_3 \dots$$

Then the output of the actor is the concatenation of the results of a sequence of applications of the firing function:

$$\begin{aligned} F_0(s) &= \perp_n \\ F_1(s) &= (\phi(F_0))(s) = f(u_1) \\ F_2(s) &= (\phi(F_1))(s) = f(u_1) \cdot f(u_2) \\ &\dots \end{aligned}$$

The problem we address now is *scheduling*: how to choose which actor to fire when there are choices.



## Adapting Parks' Strategy to Dataflow

- Require that the scheduler “know” how many tokens a firing will produce on each output port before that firing is invoked.
- Start with an arbitrary bound on the capacity of all buffers.
- Execute enabled actors that will not overflow the buffers on their outputs.
- If deadlock occurs and at least one actor is blocked on an enabled, increase the capacity of at least one buffer to allow an actor to fire.
- Continue executing, repeatedly checking for deadlock.

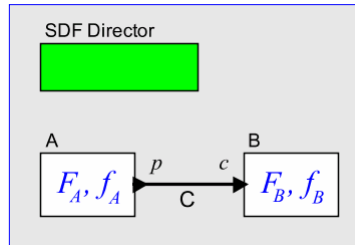
Lee 16: 5

## But Often the Firing Sequence can be Statically Determined! A History of Attempts:

- Computation graphs [Karp & Miller - 1966]
- Process networks [Kahn - 1974]
- Static dataflow [Dennis - 1974]
- Dynamic dataflow [Arvind, 1981]
- K-bounded loops [Culler, 1986]
- Synchronous dataflow [Lee & Messerschmitt, 1986] **today**
- Structured dataflow [Kodosky, 1986]
- PGM: Processing Graph Method [Kaplan, 1987]
- Synchronous languages [Lustre, Signal, 1980's]
- Well-behaved dataflow [Gao, 1992]
- Boolean dataflow [Buck and Lee, 1993]
- Multidimensional SDF [Lee, 1993]
- Cyclo-static dataflow [Lauwereins, 1994]
- Integer dataflow [Buck, 1994]
- Bounded dynamic dataflow [Lee and Parks, 1995]
- Heterochronous dataflow [Girault, Lee, & Lee, 1997]
- Parameterized dataflow [Bhattacharya and Bhattacharyya 2001]
- ...

Lee 16: 6

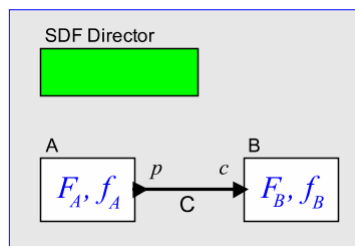
## Statically Schedulable Dataflow – SSDF Historically called: Synchronous Dataflow (SDF)



If the number of tokens consumed and produced by the firing of an actor is constant, then static analysis can tell us whether we can schedule the firings to get a useful execution, and if so, then a finite representation of a schedule for such an execution can be created.

Lee 16: 7

## Balance Equations



Let  $q_A, q_B$  be the number of firings of actors A and B.

Let  $p_C, c_C$  be the number of token produced and consumed on a connection C.

Then the system is *in balance* if for all connections C

$$q_A p_C = q_B c_C$$

where A produces tokens on C and B consumes them.

Lee 16: 8

## Relating to Infinite Firings

Of course, if  $q_A = q_B = \infty$ , then the balance equations are trivially satisfied.

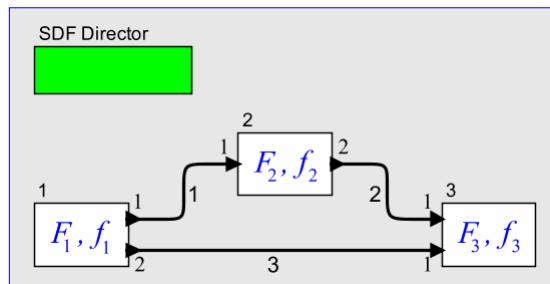
By keeping a system in balance as an infinite execution proceeds, we can keep the buffers bounded.

Whether we can have a bounded infinite execution turns out to be decidable for SSDF models.

Lee 16: 9

## Example

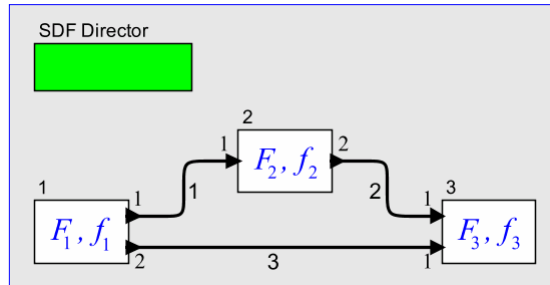
Consider this example, where actors and arcs are numbered:



The balance equations imply that actor 3 must fire twice as often as the other two actors.

Lee 16: 10

## Compactly Representing the Balance Equations



production/consumption matrix

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$

Actor 1

Connector 1

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

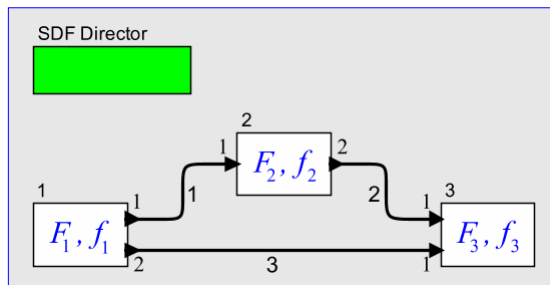
firing vector

balance equations

$$\Gamma q = \vec{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Lee 16: 11

## Example



A solution to balance equations:

$$q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

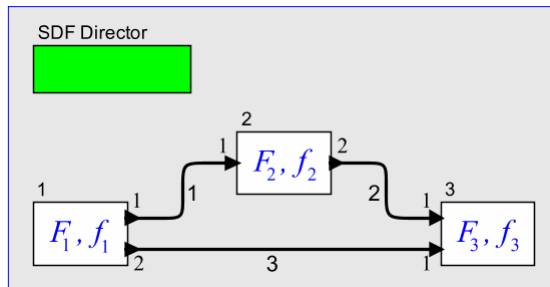
$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$

$$\Gamma q = \vec{0}$$

This tells us that actor 3 must fire twice as often as actors 1 and 2.

Lee 16: 12

## Example



But there are many solutions to the balance equations:

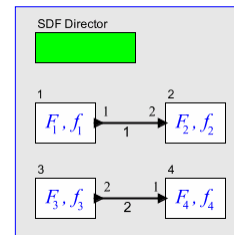
$$q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \quad q = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad q = \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \quad q = \begin{bmatrix} -1 \\ -1 \\ -2 \end{bmatrix} \quad q = \begin{bmatrix} \pi \\ \pi \\ 2\pi \end{bmatrix} \quad \Gamma q = \vec{0}$$

We will see that for “well-behaved” models, there is a unique least positive solution.

Lee 16: 13

## Disconnected Models

For a disconnected model with two connected components, solutions to the balance equations have the form:



Solutions are linear combinations of the solutions for each connected component:

$$\Gamma = \begin{bmatrix} 1 & -2 & 0 & 0 \\ 0 & 0 & 2 & -1 \end{bmatrix} \quad q = \begin{bmatrix} 2n \\ n \\ m \\ 2m \end{bmatrix} = n \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix} + m \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \end{bmatrix} \quad \Gamma q = \vec{0}$$

Lee 16: 14

## Disconnected Models are Just Separate Connected Models



Define a *connected model* to be one where there is a path from any actor to any other actor, and where every connection along the path has production and consumption numbers greater than zero.

It is sufficient to consider only connected models, since disconnected models are disjoint unions of connected models. A schedule for a disconnected model is an arbitrary interleaving of schedules for the connected components.

Lee 16: 15

## Least Positive Solution to the Balance Equations

Note that if  $p_C, c_C$ , the number of tokens produced and consumed on a connection  $C$ , are non-negative integers, then the balance equation,

$$q_A p_C = q_B c_C$$

implies:

- $q_A$  is rational if and only if  $q_B$  is rational.
- $q_A$  is positive if and only if  $q_B$  is positive.

Consequence: Within any connected component, if there is any solution to the balance equations, then there is a unique least positive solution.

Lee 16: 16

## Rank of a Matrix

The rank of a matrix  $\Gamma$  is the number of linearly independent rows or columns. The equation

$$\Gamma q = \vec{0}$$

is forming a linear combination of the columns of  $G$ . Such a linear combination can only yield the zero vector if the columns are linearly dependent (this is what it means to be linearly dependent).

If  $\Gamma$  has  $a$  rows and  $b$  columns, the rank cannot exceed  $\min(a, b)$ . If the columns or rows of  $\Gamma$  are re-ordered, the resulting matrix has the same rank as  $\Gamma$ .

Lee 16: 17

## Rank of the Production/Consumption Matrix

Let  $a$  be the number of actors in a connected graph. Then the *rank* of the production/consumption matrix  $\Gamma$  must be  $a$  or  $a - 1$ .

$\Gamma$  has  $a$  columns and at least  $a - 1$  rows. If it has only  $a - 1$  columns, then it cannot have rank  $a$ .

If the model is a *spanning tree* (meaning that there are barely enough connections to make it connected) then  $\Gamma$  has  $a$  rows and  $a - 1$  columns. Its rank is  $a - 1$ . (Prove by induction).

Lee 16: 18



## Dynamics of Execution

Consider a model with 3 actors. Let the *schedule* be a sequence  $v : N_0 \rightarrow B^3$  where  $B = \{0, 1\}$  is the binary set. That is,

$$v(n) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

to indicate firing of actor 1, 2, or 3.

Lee 16: 21

## Buffer Sizes and Periodic Admissible Sequential Schedules (PASS)

Assume there are  $m$  connections and let  $b : N_0 \rightarrow N^m$  indicate the buffer sizes prior to the each firing. That is,  $b(0)$  gives the initial number of tokens in each buffer,  $b(1)$  gives the number after the first firing, etc. Then

$$b(n+1) = b(n) + \Gamma v(n)$$

A *periodic admissible sequential schedule (PASS)* of length  $K$  is a sequence

$$v(0) \dots v(K-1)$$

such that  $b(n) \geq \vec{0}$  for each  $n \in \{0, \dots, K-1\}$ , and

$$b(K) = b(0) + \Gamma[v(0) + \dots + v(K-1)] = b(0)$$

Lee 16: 22

## Periodic Admissible Sequential Schedules

Let  $q = v(0) + \dots + v(K-1)$   
and note that we require that  $\Gamma q = \vec{0}$ .

A PASS will bring the model back to its initial state, and hence it can be repeated indefinitely with bounded memory requires.

A necessary condition for the existence of a PASS is that the balance equations have a non-zero solution. Hence, a PASS can only exist for a consistent model.

Lee 16: 23



## SSDF Theorem 1

We have proved:

For a connected SSDF model with  $a$  actors, a *necessary* condition for the existence of a PASS is that the model be consistent.

Lee 16: 24



## SSDF Theorem 2

We have also proved:

For a consistent connected SSDF model with production/consumption matrix  $\Gamma$ , we can find an integer vector  $q$  where every element is greater than zero such that

$$\Gamma q = \vec{0}$$

Furthermore, there is a unique least such vector  $q$ .

Lee 16: 25



## SSDF Sequential Scheduling Algorithms

Given a consistent connected SSDF model with production/consumption matrix  $\Gamma$ , find the least positive integer vector  $q$  such that  $\Gamma q = \vec{0}$ .

Let  $K = \mathbf{1}^T q$ , where  $\mathbf{1}^T$  is a row vector filled with ones. Then for each of  $n \in \{0, \dots, K-1\}$ , choose a firing vector

$$v(n) = \left\{ \begin{bmatrix} 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \dots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ \dots \\ 1 \end{bmatrix} \right\}$$

The number of rows in  $v(n)$  is  $a$ .

Lee 16: 26

## SSDF Sequential Scheduling Algorithms (Continued)

.. such that  $b(n+1) = b(n) + \Gamma v(n) \geq \vec{0}$  (each element is non-negative), where  $b(0)$  is the initial state of the buffers, and

$$\sum_{n=0}^{K-1} v(n) = q$$

The resulting *schedule* ( $v(0), v(1), \dots, v(K-1)$ ) forms one cycle of an infinite periodic schedule.

Such an algorithm is called an *SSDF Sequential Scheduling Algorithm (SSSA)*.

Lee 16: 27

## SSDF Theorem 3

If an SSDF model has a correct infinite sequential execution that executes in bounded memory, then any SSSA will find a schedule that provides such an execution.

Proof outline: Must show that if an SSDF has a correct, infinite, bounded execution, then it has a PASS of length  $K$ . See Lee & Messerschmit [1987]. Then must show that the schedule yielded by an SSSA is correct, infinite, and bounded (trivial).

Note that every SSSA terminates.

Lee 16: 28

## Creating a Scheduler

Given a connected SSDF model with actors  $A_1, \dots, A_a$ :

Step 1: Solve for a rational  $q$ . To do this, first let  $q_1 = 1$ . Then for each actor  $A_i$  connected to  $A_1$ , let  $q_i = q_1 m/n$ , where  $m$  is the number of tokens  $A_1$  produces or consumes on the connection to  $A_i$ , and  $n$  is the number of tokens  $A_i$  produces or consumes on the connection to  $A_1$ . Repeat this for each actor  $A_j$  connected to  $A_i$  for which we have not already assigned a value to  $q_j$ . When all actors have been assigned a value  $q_j$ , then we have found a rational vector  $q$  such that  $\Gamma q = \vec{0}$ .

Lee 16: 29

## Creating a Scheduler (continued)

Step 2: Solve for the least integer  $q$ . Use Euclid's algorithm to find the least common multiple of the denominators for the elements of the rational vector  $q$ . Then multiply through by that least common multiple to obtain the least positive integer vector  $q$  such that

$$\Gamma q = \vec{0}$$

Let  $K = \mathbf{1}^T q$ .

Lee 16: 30

## Creating a Scheduler (continued)

Step 3: For each  $n \in \{0, \dots, K-1\}$ :

1. Given buffer sizes  $b(n)$ , determine which actors have firing rules that are satisfied (every source actor will have such a firing rule).
2. Select one of these actors that has not already been fired the number of times given by  $q$ . Let  $v(n)$  be a vector with all zeros except in the position of the chosen actor, where its value is 1.
3. Update the buffer sizes:

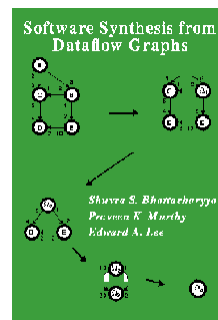
$$b(n+1) = b(n) + \Gamma v(n)$$

Lee 16: 31

## A Key Question: If More Than One Actor is Fireable in Step 2, How do I Select One?

Optimization criteria that might be applied:

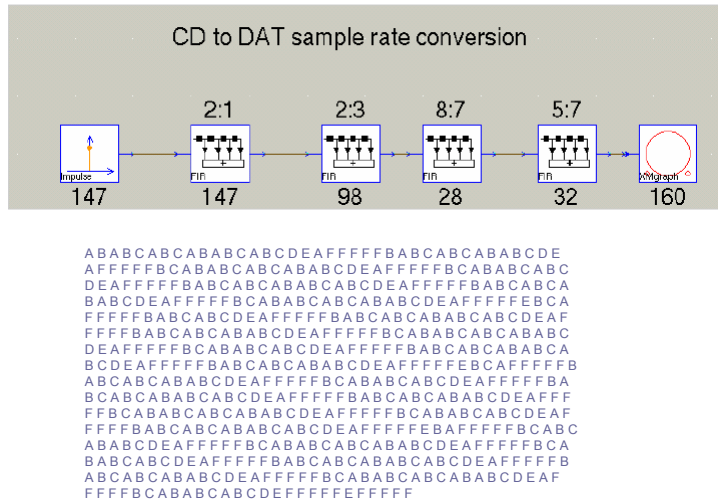
- Minimize buffer sizes.
- Minimize the number of actor activations.
- Minimize the size of the representation of the schedule (code size).



See S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*, Kluwer Academic Press, 1996.

Lee 16: 32

# Minimum Buffer Schedule



Source: Shuvra Bhattacharyya

Lee 16: 33

# Code Generation (Circa 1992)

Block specification for DSP code generation in Ptolemy Classic:

```

codeblock(std) {
  : initialize address registers for coef and
  delayLineove #Saddr(coef)+$val(coefLen)-1,r3
: insert here
  move $ref(delayLineStart),r5
: delayLine
  move #val(stepSize),x1
  move $ref(error),x0
  mpyr x0,x1,a
  move a,x0
  move x:(r3),b y:(r5)+.y0
}

codeblock(loop) {
  do #val(loopVal), $label(endloop)
  macr x0,y0,b
  move b,x:(r3)-
  move x:(r3),b y:(r5)+.y0
$label(endloop)
}

codeblock(noloop) {
  macr x0,y0,b
  move b,x:(r3)-
  move x:(r3),b y:(r5)+.y0
}

```

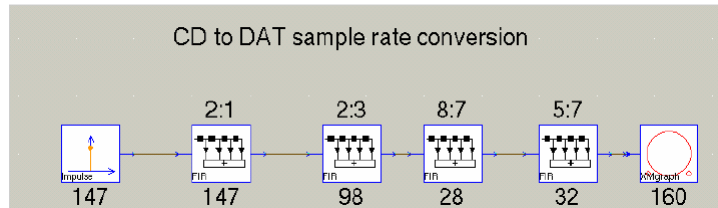
macros defined by the code generator

alternative code blocks chosen based on parameter values

Lee 16: 34

## Scheduling Tradeoffs

(Bhattacharyya, Parks, Pino)



Scheduling strategy	Code	Data
Minimum buffer schedule, no looping	13735	32
Minimum buffer schedule, with looping	9400	32
Worst minimum code size schedule	170	1021
Best minimum code size schedule	170	264

Source: Shuvra Bhattacharyya

Lee 16: 35

## Parallel Scheduling

It is easy to create an SSSA that as it produces a PASS, it constructs an *acyclic precedence graph* (APG) that represents the dependencies that an actor firing has on prior actor firings.

Given such an APG, the parallel scheduling problem is a standard one where there are many variants of the optimization criteria and scheduling heuristics.

See many papers on the subject on the Ptolemy website.

Lee 16: 36

## Conclusions and Open Issues

- SSDF models have actors that produce and consume a fixed (constant) number of tokens on each arc.
- A periodic admissible sequential schedule (PASS) is a finite sequence of firings that brings buffers back to their initial state and keeps buffer sizes non-negative.
- A necessary condition for the existence of a PASS is that the balance equations have a non-trivial solution.
- A class of algorithms has been identified that will always find a PASS if one exists.