ECE 587 – Hardware/Software Co-Design
Lecture 06 Process-Based Models II

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

February 3, 2025

Reading Assignment

- This lecture: 3.1.2, 3.2
- Next two lectures: Concurrency in Practice

Outline

Synchronous Data Flow (SDF)

- Actor model restricts how processes execute.
- Further restriction on how actors consume and product tokens.
  - Fire (execute) an actor: actors only perform substantial amount of computation only when enough tokens are received.
  - Once fired, an actor will generate enough tokens.
  - How many are enough?
- Synchronous Data Flow (SDF)
  - Each actor consumes and produces a predefined number of tokens per channel, per firing.
  - The numbers could be different for different channels, but remain fixed for a particular channel and read/write operation.

SDF Example

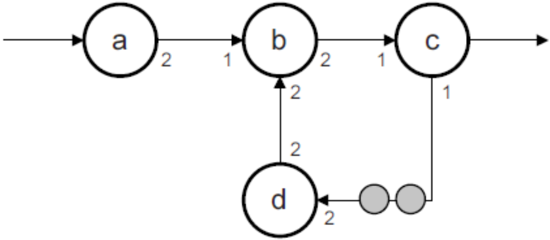


FIGURE 3.2 Synchronous Data Flow (SDF) example

(Gajski et al., 2009)

Actor Scheduling to Prevent Deadlocks

- Can we prevent deadlock?
  - Not for all SDFs, but we can decide if there will be a deadlock.
  - Deadlock only happens when there is not enough tokens on certain arcs.
- Assume initially there are enough tokens on each arc.
  - So we may freely fire any actor for certain amount of times.
- If we can find a scheduling to fire the actors such that the number of tokens will remain unchanged afterwards, then we know there is no deadlock if we repeat such scheduling.
  - Also known as the relative execution rates of actors.

Determine the Relative Execution Rates I

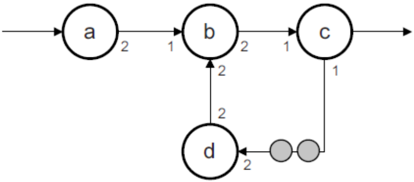


FIGURE 3.2 Synchronous Data Flow (SDF) example

(Gajski et al., 2009)

- Assigning each process an unknown representing the times the actor should be fired.
- For each arc, writing down an equation requiring the number of produced tokens to be equal to the consumed tokens.

Determine the Relative Execution Rates II

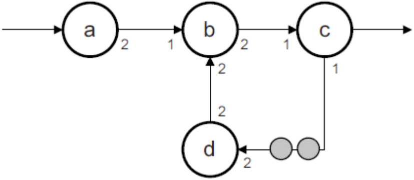


FIGURE 3.2 Synchronous Data Flow (SDF) example

(Gajski et al., 2009)

$$
\begin{aligned}
2A &= B \\
2B &= C \\
C &= 2D \\
2D &= 2B
\end{aligned}
$$

- Now you have a system of linear equations.
  - Though there are usually more equations (number of arcs) than unknowns (number of nodes).
  - 0 is a solution but we are looking for a solution other than that.

Determine the Relative Execution Rates III

$$\begin{aligned} 2A &= B \\ 2B &= C \\ C &= 2D \\ 2D &= 2B \end{aligned}$$

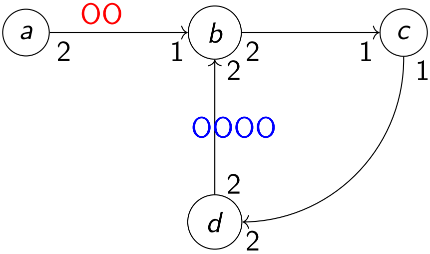
- $A = 1, B = 2, C = 4, D = 2$
  - If no such solution exists, we know deadlock or overflow eventually happens.
  - Otherwise, we can find a minimum integer solution as the relative execution rates.
- Assumptions: for each channel
  - There will be enough tokens initially.
  - There will be enough buffer to hold all intermediate tokens.
  - But how many are enough?

Implementation Considerations for SDF

- Our scheduling as relative execution rates doesn't specify the order to fire those actors.
  - This leaves great flexibility in determining an order of firing.
- Different orders may require different resources to complete one round of firings.
  - Number of initial tokens on each arc
  - Sizes of the queues
- Tools may help you and it could even be possible to share the memory for queues.

SDF Scheduling Example I

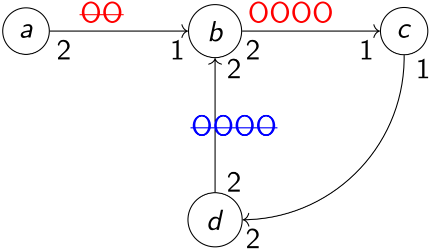
- Consider a simple repeating schedule 'abbccccdd'
  - Recall $A = 1, B = 2, C = 4, D = 2$
- After firing 'a',



- To fire 'bb', we need 4 initial tokens on the d to b edge.

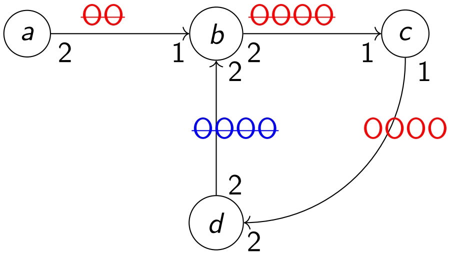
SDF Scheduling Example I (Cont.)

- After firing 'bb',



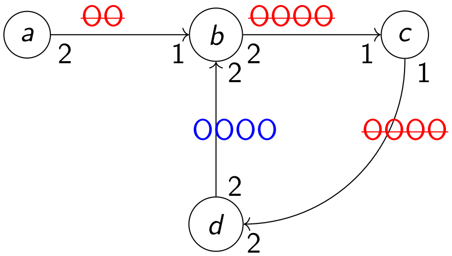
SDF Scheduling Example I (Cont.)

- After firing 'cccc',



SDF Scheduling Example I (Done)

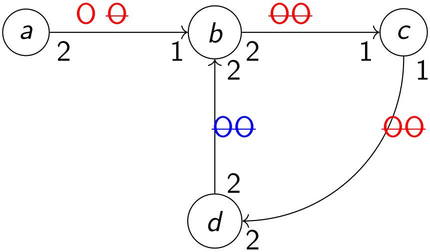
- After firing 'dd',



- We have returned to the initial setting where there are 4 tokens on the d to b edge.
- We need storage to store 14 tokens.

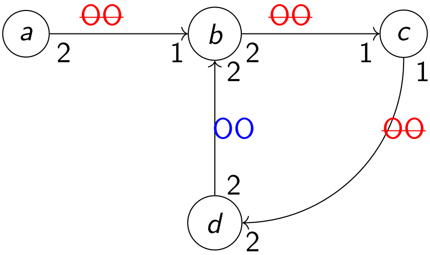
SDF Scheduling Example II

- What if the schedule is 'abccdbccd'?
  - To fire 'b' after 'a', we need 2 initial tokens on the d to b edge.
- After firing 'abccd',



SDF Scheduling Example II (Done)

- After firing 'bccd',



- We have returned to the initial setting where there are 2 tokens on the d to b edge.
- We need storage to store 8 tokens.
- This is a better schedule!

Outline

Synchronous Data Flow (SDF)

Data Flow Graphs

System Design Languages

Data Flow Graphs (DFG)

- Could be treated as a special SDF
  - Each actor consumes/produces 1 token from every input/to every output.
  - No cycle: actors are fired in their topological order once to compute a set of output from a set of input.
- A set of statements without branches can be transformed into a DFG.
  - E.g. a basic block (BB)
  - Except inputs and outputs, variables may be eliminated.
  - Ordering of operations may be relaxed.
  - Further compiling to certain target instruction set could be viewed as a scheduling of the processes/actors on a single processor.
- The actors should match your desired level of abstraction.
- Parallelism can be exploited by firing the actors according to their levels.

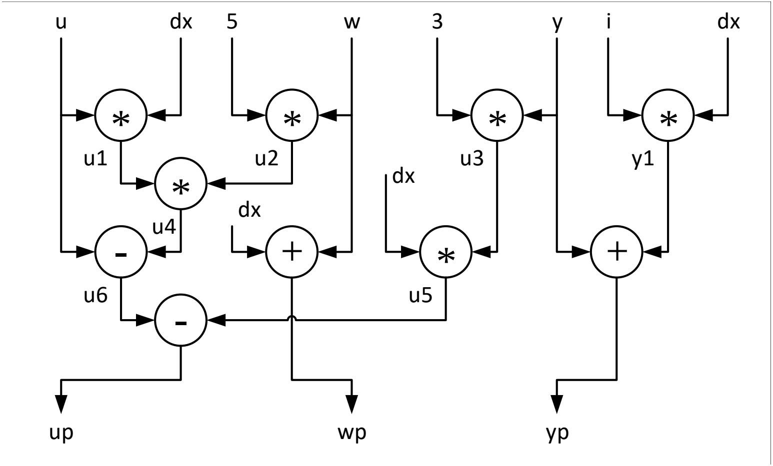
Example Statements

```
inputs: u, w, y, dx, i
outputs: up, wp, yp
temporary variables: u1, u2, u3, u4, u5, u6, y1

  u1 = u *dx;
  u2 = 5 *w;
  u3 = 3 *y;
  y1 = i *dx;
  wp = w +dx;
  u4 = u1*u2;
  u5 = dx*u3;
  yp = y +y1;
  u6 = u -u4;
  up = u6-u5;
```

- Such code may appear in a loop that need to be optimized.
  - `u, w, y` may be updated to `up, wp, yp` respectively at the end of the iteration.
- Static Single Assignment Form (SSA): each variable is assigned once since othrewise they can be renamed.

DFG Example



Outline

System Design Languages

- Specify system functionality with minimum design effort by capturing models.
  - Allow further automated processings/transformations, e.g. simulation/debugging, synthesis/optimization, verification.
  - Natural languages are not a good choice as they are ambiguous and incomplete.
- What language is ideal for system design?
- Other relevant questions
  - Why there are so many languages?
  - Which one should I learn?
  - How should I learn a particular language?

Typical Languages

- C/C++/Java
  - The core language supports sequential programs and various abstraction mechanisms, e.g. OOP, for user-defined libraries.
  - User-defined libraries cover many application domains.
- Structural Verilog/VHDL
  - Support mapping to hardware via a structural model, i.e. components and interconnects.
  - Primarily targeted at RTL designs.
  - Extensions support behavioral models, i.e. processes and sequential programs, though results when synthesized into hardware may be inferior.
- Matlab
  - For matrix computations.
  - Sequential programs are also supported, though the performance will be inferior if the computation could be written in matrix forms.

Domain Specific Languages

- Languages are created to
  - Solve problems in specific application domains.
  - Provide abstractions so that users can adopt them for specific application domains via building their own libraries.
- Application domains are distinguished by their respective models.
- To learn a new language or a new library written in a language you are familiar with,
  - Learn the models specific to the associated application domain.
  - Learn how to capture the models using the language/library.

Choose a System Design Language

- Ideal system design languages should support models used to specify the whole system at all abstraction levels.
  - Can we design one?
- Practical considerations
  - Labor force: can you motivate the designers to learn this new language?
  - Legacy code: can you persuade the industry to move their code to this new language?
- Practical system design languages
  - Extend existing languages to support system designs
  - Different trade-offs leads to different choices of base languages.
  - Modern tools can support inter-operation of multiple system design languages.

Summary

- By restricting how processes execute and how communications happen, we may obtain models that guarantee no-deadlock while still flexible enough for different implementations.
- Sequential programs are state-based, though we may extract parallelism through its data flow.
- There are no ideal system design languages.