Dataflow Models

1. Dataflow Models: an Example

2. Kahn Process Networks: a Deterministic Model

3. Synchronous Dataflow: Statically Schedulable Dataflow Models

4. Deriving a static Schedule for Synchronous Dataflow Models

Dataflow Models (cont'd)

Dataflow models are suitable for signal-processing algorithms:
- Code/decode, filter, compression, etc.
- Streams of periodic and regular data samples
- Typically signal-processing algorithms are expressed as block diagrams; this naturally fits to dataflow semantics.

Dataflow Models (cont'd)

Systems are specified as directed graphs where:
- nodes represent computations (processes);
- arcs represent totally ordered sequences (streams) of data (tokens).

Depending on their particular semantics, several models of computation based on dataflow have been defined:
- Kahn process networks
- Dataflow process networks
- Synchronous dataflow

Typical case of data-driven concurrency (see Fö3, slide 14).

The internal computation of a process can be specified in any programming language (e.g. C). This is called the host language.
Kahn Process Networks (KPN)

- Processes communicate by passing data tokens through unidirectional FIFO channels.
- Writes to the channel are non-blocking.
- Reads are blocking:
  - the process is blocked until there is sufficient data in the channel

A process that tries to read from an empty channel waits until data is available. It cannot ask whether data is available before reading and, for example, if there is no data, decide not to read that channel.

Kahn Process Networks (cont'd)

Kahn process networks are deterministic:

- For a certain sequence of inputs, there is only one possible sequence of outputs (regardless, for example, how long time it takes for a certain computation or communication to finish).

Looking only at the specification (and not knowing anything about implementation) you can exactly derive the output sequence corresponding to a certain input sequence.

Kahn Process Networks: a Simpler Example

Process `p1`( in int a, out int x, out int y) {
  int k;
  loop
    k = a.receive();
    if k mod 2 = 0 then
      x.send(k);
    else
      y.send(k);
    end if;
  end loop;
}

Process `p2`( in int a, out int x) {
  int k;
  loop
    k = a.receive();
    x.send(k);
  end loop;
}

Process `p3`( in int a, in int b, out int x) {
  int k; bool sw = true;
  loop
    if sw then
      k = a.receive();
    else
      k = b.receive();
    end if;
    x.send(k);
    sw = !sw;
  end loop;
}

channel int I, O, C1, C2, C3, C4;
`p1`(I, C1, C2);
`p2`(C1, C3);
p2(C2, C4);
p3(C3, C4, O);
Process q3\(\text{int a, in b, out int x) { int } k; \text{bool sw = true; loop;}
\begin{align*}
&\text{if sw then } k = a.\text{receive}() \text{ on timeout(d) do select on a,b} \\
&\text{or } b: k = b.\text{receive}(); \text{ end select;}
\end{align*}
\begin{align*}
&\text{else } k = b.\text{receive}() \text{ on timeout(d) do select on a,b} \\
&\text{or } a: k = a.\text{receive}(); \text{ end select;}
\end{align*}
\begin{align*}
&\text{end if}; x.\text{send}(k); \\
&\text{sw = !sw;}
\end{align*}
\begin{align*}
&\text{end loop; } \}
\end{align*}

Consider q3 instead of p3:
• Process q3 first tries channel a or b, depending on sw, like in the previous version.
• But, instead of blocking, if nothing comes after a timeout period d, q3 will read from any of the two channels, taking the token which is available first.

With q3 we do not have a Kahn process network
• The system is not deterministic.

### Conclusions
- With an implementation such that channel C3 is very fast and C4 is very slow.
- With an implementation such that channel C3 is very slow and C4 is very fast.

### Scheduling of Kahn Process Networks
Let us come back to the Kahn process network (slide 8):
- Let us imagine we have to implement the system on a single processor architecture.
- Let's try the following static schedule:

#### The system will block!
- Example: First token is 4; p1 passes to C1; p2; passes to C3; p2; waits for ever.

### Scheduling of Kahn Process Networks (cont'd)
Let's see some other schedules:
- The system will block!
  - Example: first token 4.
- The system will block!
  - Example: first token 4.
- The system will block!
  - Example: sequence starting with 4, 8.
Scheduling of Kahn Process Networks (cont'd)

- Kahn process networks are dynamic dataflow models: their behavior is data dependent; depending on the input data one or the other process is activated.
- Kahn process networks cannot be scheduled statically ⇒ It is not possible to derive, at compile time, a sequence of process activations such that the system does not block under any circumstances.

Kahn process networks have to be scheduled dynamically ⇒ which process to activate at a certain moment has to be decided, during execution time, based on the current situation.

There is an overhead in implementing Kahn process networks.

Another problem: memory overhead with buffers. Potentially, it is possible that the memory need for buffers grows unlimited (see channel C4 on slide 8).

Kahn process networks are strong in their expressive power but sometimes cannot be implemented efficiently.

Introduce limitations so that you can get efficient implementations.

Synchronous Dataflow Models

- Dataflow process networks are a particular case of Kahn process networks.
  - A particular kind of dataflow process networks, which can be efficiently implemented, are synchronous dataflow (SDF) networks.

- Synchronous dataflow networks are Kahn process networks with additional restriction:
  - At each activation (firing) a process produces and consumes a fixed number of data tokens on each of its outgoing and incoming channels respectively.
  - For a process to fire, it must have at least as many tokens on its input channels as it has to consume.

Synchronous dataflow models are less expressive than Kahn process networks:
- With SDF models it is impossible to express conditional firing, where a process' firing depends on a certain condition (e.g. processes p1 and p3 in slide 8); SDF are static dataflow models.

For the above reduced expressiveness, however, we get two nice features of SDF models:
1. Possibility to produce static schedules.
2. Limited and predictable amount of needed buffer space.
Synchronous Dataflow Models (cont’d)

- Arrows are marked with the number of tokens produced or consumed.
- This is a simple “single-rate” system: every process is activated one single time before the system returns to its initial state.
- The same is the case for the example used in Fö 1 & 2 (slide 18).

Possible static schedule:

```
A B C D
```

Deriving a static schedule for SDF

- For a correct synchronous dataflow network there exists a sequence of firings which returns the network in its original state. This sequence represents a static schedule which has to be repeated in a cycle.
- The schedule is such that a finite amount of memory is required (no infinite buffers)

Problem

How to derive such a cyclic schedule?

Deriving a static schedule for SDF (cont’d)

- Along the periodic sequence of firing, on each arc the same number of tokens has to be produced and consumed.
- \(a, b, c, d\): the number of firings, during a period, for process A, B, C, D.

Balance equations:

\[
\begin{align*}
2a - 4b &= 0 \\
2b - 2c &= 0 \\
2c - d &= 0 \\
2b - 2d &= 0 \\
2d - a &= 0 \\
\end{align*}
\]

\[
\begin{bmatrix}
2 & -4 & 0 & 0 \\
0 & 1 & -2 & 0 \\
0 & 0 & 2 & -1 \\
0 & 2 & 0 & -2 \\
-1 & 0 & 0 & 2 \\
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d \\
\end{bmatrix}
= 0
\]
Deriving a static schedule for SDF (cont'd)

For a certain SDF network (graph) we get an equation:

\[ \Gamma q = 0 \]

If there is no \( q \neq 0 \) which satisfies the equation above \( \Rightarrow \) there is no static schedule (there is a rate inconsistency between processes).

For our SDF graph, this solution is: \( a=4, b=2, c=1, d=2 \).

• The numbers above are telling us how often each task is activated during one period.
• Based on these numbers a periodic static schedule can be elaborated.

Buffer space needed:
- A-B: 6; B-C: 4; A-C: 3;
- Total: 13 if buffers not shared
  9 if buffers shared

Buffer space needed:
- A-B: 4; B-C: 4; A-C: 3;
- Total: 11 if buffers not shared
  8 if buffers shared
Deriving a static schedule for SDF (cont’d)

With this example we have a rate inconsistency ⇒ No static, periodic schedule with finite buffers is possible.

- There is no solution for the equation, different from $a=b=c=0$.
- It is easy to observe that on the arc $A \rightarrow C$, tokens continuously accumulate.

```
|   -1 0
| 0 1 -1
| 0 0 -1
```

Treatment of Time

Dataflow systems are asynchronous concurrent.

- Events can happen at any time.
- There exists a partial order of events:
  - Producing a token by A strictly precedes consuming a token by B.
  - There is no order between consuming a token by B and consuming a token by C.

Summary

- Dataflow models consist of nodes representing computation and arcs representing totally ordered sequences of data. They are particularly suitable for signal-processing applications and, in general, applications dealing with streams of periodic/regular data samples.
- Several models of computation based on dataflow have been defined. They represent different trade-offs between expressiveness, on the one side, and determinism or potential of efficient implementation, on the other side.
- Kahn Process Networks: FIFO channels and blocking read. They are deterministic: For a certain sequence of inputs, there is only one possible sequence of outputs. Kahn Process Networks, in general, cannot be scheduled statically.

Summary (cont’d)

- Synchronous Dataflow Networks introduce an additional restriction: at each activation a process produces and consumes a fixed number of tokens.
- For a correct Synchronous Dataflow Networks a static schedule can be derived.
- Dataflow models are asynchronous concurrent. Events can happen at any time. There exists a partial order of events.