Overview of Dataflow Languages

Waheed Ahmad
w.ahmad@utwente.nl
• “The purpose of models is not to fit the data but to sharpen the questions.”
  Samuel Karlins
  11th R.A Fisher Memorial Lecture
  Royal Society of London, 1983

• In this presentation, we will
  – introduce ourselves with concept of Dataflow Languages
  – examine existing models; Kahn Process Networks and Synchronous Dataflow (SDF)
  – get familiar how new extension of SDF i.e. Scenario-Aware Dataflow overcomes bottlenecks of previous models.
# Table of Contents

1. Need for Dataflow Languages
2. Dataflow Languages
   i. Characteristics of Dataflow Languages
   ii. Benefits of Dataflow Language
3. Kahn Process Networks
   i. Definition
   ii. Properties i.e. Monotonic Mapping and Determinism
   iii. Limitations
4. Synchronous Dataflow
   i. Definition
   ii. Firing Rules and Scheduling
   iii. Limitations
5. Scenario-Aware Dataflow
   i. Definition and Semantics
   ii. SDF vs. SADF
   iii. Example – MP3 Decoder
6. Conclusions
(1) Why Dataflow Language?

• To formalise the model of a system.
• To be able to formally synthesise, analyse, verify qualitative and quantities properties of system.
  – Throughput Analysis
  – Latency Analysis
  – Buffer Analysis
  – Boundedness
  – Deadlock Absence
• Solid mathematical foundation.
(2) Dataflow Languages

- Consists of directed graph of the data flowing between operations.
- Each program instruction is represented by a node on the graph.
- Inputs and outputs of each instruction are denoted by arcs to and from each node.
- The arcs carry data from one node to another, providing for the flow of data throughout the program.

\[ A := X + Y \]
\[ B := Y / 10 \]
\[ C := A \times B \]
(2) Basic Characteristics

• Every process runs concurrently.
• Processes can be described with imperative code.
• Processes communicating through FIFO buffers.
• FIFO buffer helps to decouple source and sink.
  – Therefore source process can continue to produce data without having to wait for the sink to finish processing the previous data
(2) Benefits

• Ability to achieve massive parallelism.
  – Each instruction within a dataflow program can be executed whenever data is waiting on its incoming arcs.
  – It is safe to execute multiple instructions simultaneously without the instructions affecting each other.

• Representation with nodes and arcs over imperative programming languages: The ability to develop programs graphically.
(3) Kahn Process Networks (KPN)

- Proposed by Kahn in 1974 as a general-purpose scheme for parallel programming
  - Concurrent processes communicate via infinitely large unidirectional FIFO channels.
  - Language augmented with send() and wait() operations that write and read from channels
    - read: destructive and blocking
      - reading an empty channel blocks until data is available
    - write: non-blocking
- Laid the theoretical foundation for dataflow.
- Unique attribute: deterministic
(3) A Kahn Process

- From Kahn’s original 1974 paper

```c
process f(in int u, in int v, out int w)
{
    int i; bool b = true;
    for (;;) {
        i = b ? wait(u) : wait(w);
        printf("%i\n", i);
        send(i, w);
        b = !b;
    }
}
```

Process alternately reads from u and v, prints the data value, and writes it to w
(3) Monotonic Mapping

• Any process P in Kahn Process Networks with m inputs and n outputs is a mapping from its input signals to its output signals, \( P : S^m \rightarrow S^n \).

• A process is monotonic in its mapping from input signals to output signals if

\[
    s_1 \sqsubseteq s_2 \implies P(s_1) \sqsubseteq P(s_2)
\]

• Networks of monotonic processes are always determinate.
(3) Determinism

- A Kahn process uses blocking read, making it impossible to test the presence of input data *without consuming them*.
- If a process tries to read from an empty input it is suspended until it has enough input data.
- A process may not “test” channel for presence of data.
- At any given point a process is either “enabled” or “blocked” waiting for data on one of its channels.

*Fig. 1. This process network does not specify the relative timing of the processing in nodes B and C. If D is a nondeterminate merge, it does not specify in which order the results should appear at E.*
(3) Kahn Process - Limitations

• Difficult to schedule.
  – Schedule is a finite sequence of node firings which eventually leads back to the initial state of the graph.

• Might be appropriate for coarse-grain systems.

• Evaluating correctness and performance is undecidable.
(4) Synchronous Dataflow (SDF)

• Proposed by Edward Lee and David Messerchmitt, Berkeley, 1987
• Synchronous Data Flow is a special case of Data Flow in which number of input samples consumed on each input and the number of output samples produced on each output can be specified a priori.
(4) Synchronous Dataflow (SDF)

- A SDF graph consists of nodes and arcs.
- Nodes are called *actors* and they represent operations.
- Arcs represent data values called *tokens* which stored in first-in first-out (FIFO) queues.
- The word token represents any data type or any data structure.
- Nodes are free from side effects.
(4) Firing Rules

- An actor is enabled for execution when tokens equal to its consumption rate are available on all of the inputs.
- When an actor executes, it always produces and consumes the same fixed amount of tokens.
- The flow of data through the graph may not depend on values of the data which means SDF graph may not contain data-dependent switch statements.
- Delay is a property of an arc, and a delay of $n$ samples means that $n$ tokens are initially in the queue of that arc.
(4) Iterations

- Balance equations (one for each channel):
  \[ f_A N = f_B M \]

- Schedulable statically
- Get a well-defined “iteration”
- Decidable:
  - buffer memory requirements
  - deadlock

```plaintext
fire A {
    ...
    produce N
    ...
}

fire B {
    ...
    consume M
    ...
}
```

Channel

number of tokens consumed
number of firings per “iteration”
number of tokens produced
(4) Scheduling Example

Consider the feedforward (acyclic) synchronous dataflow graph shown below:

The notation means that

- When A executes, it produces 20 tokens.
- When B executes, it consumes 10 tokens and produces 20 tokens.
- When C executes, it consumes 10 tokens.

First step in scheduling is to figure out how many times to execute each actor so that all intermediate tokens that are produced get consumed.

This is called load balancing.

In the example SDF graph above, we must

- Fire A 1 time
- Fire B 2 times
- Fire C 4 times
(4) Schedules - Example

- Dash represents Single appearance schedule in which each actor appears only once. It results in smallest code size.
- Short dash represents minimum buffer size schedule.

---

minimum buffer size
— — single appearance schedule
(4) SDF - Limitations

• Static, periodic behaviour
• Analysing only the worst-case scenario does not provide a guaranteed conservative bound on the behaviour over all scenarios.
• On the other hand, creating a single SDF graph from the worst-case execution times of actors over all scenarios, can be too pessimistic.

“Worst-case Performance Analysis of Synchronous Dataflow Scenarios” (Marc Geilen, Sander Stuijk)
(5) Scenario-Aware Dataflow

Motivation

• Goal
  – Design-time performance prediction of hard and soft real-time streaming applications running on platforms with guaranteed and/or (lossless) best-effort services.

• Starting Point
  – SDF cannot express dynamism.
  – KPN don’t support relevant analysis techniques.
(5) Scenario-Aware Dataflow

• SADF extends SDF with scenarios
  – Such scenarios denote different modes of operations in which resource requirements can differ considerably.

• SADF can express
  – Parameterised numbers of communicated tokens.
  – Inactive process
  – Discrete execution time distributions
(5) SADF Semantics

• The vertices of an SADF graph are called processes and represent pieces of behaviour.
• The edges between processes of an SADF graph are called channels
• Channels may contain an infinite amount of tokens and they forward tokens in first-in-first-out order.
• In SADF, production and consumption rates may be fixed or parameterised.
(5) SADF Semantics

- SADF distinguishes two types of processes.
  - Kernels represent the data processing behaviour of a system.
  - Detectors model the control behaviour that dynamically determines the scenarios in which the system operates.

- SADF also distinguishes two types of channels: Data channels and control channels used to exchange scenario-valued tokens.
(5) SADF Semantics

- Kernel Semantics
  - Wait for scenario-valued token on control inputs.
  - Fix scenario and hence rates and execution time distribution.

- Detector contain automata to capture occurrences of scenarios
  - Real life: data-dependent control behaviour (normal state machine)
  - Worst/best case only model: non-deterministic state machine
  - Worst/best case and average case model: Markov Chain
(5) SADF Semantics

- **Detector Semantics**
  - The firing of a detector also starts with determining the scenario.
  - If a detector has control ports, the scenario is determined by interpreting the first token on all its control ports.
  - The subscenario refers to certain consumption/production rates and an execution time distribution of the detector.
  - Together, scenario and subscenario fix the rates and execution time distribution.
  - After establishing the scenario and subscenario, the detector waits until sufficient tokens are available on its input ports.
  - Firing ends with removing the number of tokens from the (data and control) channels as specified by the consumption rates and producing the number of tokens to output ports conform the production rates.
(5) SDF vs. SADF
MPEG-4 Decoder

<table>
<thead>
<tr>
<th>Rate</th>
<th>I</th>
<th>P₀</th>
<th>Pₓ</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>c</td>
<td>99</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>99</td>
<td>0</td>
<td>x</td>
</tr>
</tbody>
</table>

x = \{30, 40, 50, 60, 70, 80, 99\}

VLD and IDCT fire per macro block
FD, MC and RC fire per frame

Throughput = 0.252525
Throughput = 0.425571

- **FD** detects frame type (9 scenarios)
  - I-frame (99 macro blocks, 0 motion vectors)
  - Pₓ-frame (x macro blocks, x motion vectors)
  - P₀-frame (still video)
(5) SADF Example – MP3 Decoder

- **MP3 Decoder**

3 Frame types
- Short (S) Frame = 96 Short Blocks
- Long (L) Frame = 32 Long Blocks
- Mixed (M) Frame = 2 Long Blocks + 90 Short Blocks

2 Block types
- Long (BL)
- Short (BS)
(5) MP3 Decoder

- **MP3 Decoder**

<table>
<thead>
<tr>
<th></th>
<th>LL</th>
<th>SS</th>
<th>LS</th>
<th>SL</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>0</td>
<td>576</td>
<td>0</td>
<td>576</td>
<td>540</td>
</tr>
<tr>
<td>f</td>
<td>576</td>
<td>0</td>
<td>576</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>576</td>
<td>576</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>h</td>
<td>576</td>
<td>0</td>
<td>0</td>
<td>576</td>
<td>540</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>L</th>
<th>S</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>576</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>576</td>
<td>540</td>
</tr>
<tr>
<td>BL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LBL</th>
<th>SBS</th>
<th>MBL</th>
<th>MBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>32</td>
<td>96</td>
<td>2</td>
<td>90</td>
</tr>
</tbody>
</table>

- **FD contains fully connected 5-state Markov chain**
- **Control tokens to RQ, RO are values L, S, M**
- **Control tokens to AR, IMDCT, FI are valued BL, BS**
- **Throughput = 2.68096e-07**
(6) Conclusions

- SADF extends SDF with scenarios
  - SADF can capture dynamism.
- SADF is fully analysable at design-time
- Parameterised numbers of communicated tokens
- Representation of inactive process
- Discrete execution time distributions