Synchronous Dataflow
A Crash Course in Infinite Programs

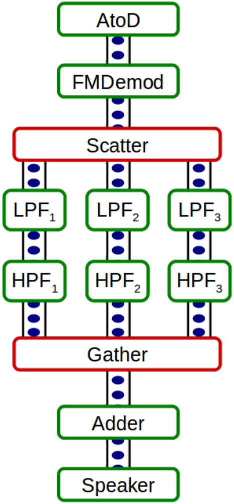
Nic Hollingum

USYD

7 Apr, 2011

Problem

- Not all programs are suited to traditional Von-Neuman model
- Digital Signal Processing
- Programming these devices is difficult[Lee 87]
- Represent this kind of computation more naturally



Solution

- Termination is ill-defined, programs can run for infinity
- Expressive notion of data rates
- Control via sequencing of data - ties inputs to outputs
- hence Synchronous Dataflow

Tokens

- Single unit of data
- Data Streams represented by sequences of tokens
- Discreet, but vary in size
- Some tokens are provided by user at execution, others arrive as input
- Data-rates of tokens must be known at compile time

Channels

- Paths along which tokens flow
- Simple FIFO buffers
- Connect data between computational units (network)

Actors

- individual computation units
  - Simple: 2 $\rightarrow$ add $\rightarrow$ 1
  - Complicated: $2^n \rightarrow$ FFT $\rightarrow 2^n$
- Consumes a number of tokens on input channels and produces on others
- ammount consumed need not equal ammoun produced by predecessor
  - Must be known at compile time
  - Possibly cosumes and produces from the same channel (feedback)
- Stateful or Stateless

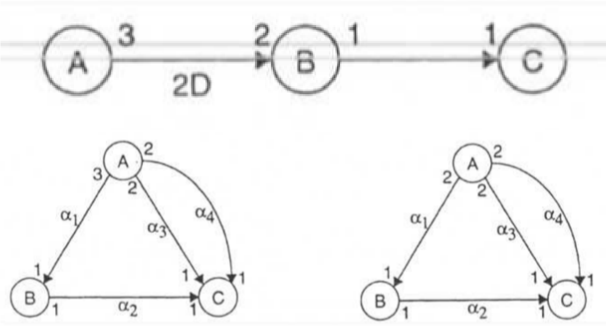
Delays

- Signal processing kinds of delays
- Allows feedback loops to be non-terminating
- Working out the required delays is possible though not easy

SDF Graphs

- Graphical representation of SDF network
- Actors are vertices
- Channels are Arcs
- Arcs are annotated with Delays and Consumption/production

SDF Graphs



[Batt 96]

Topology Matrix

- Graph: $\Gamma = \mathbb{R}^{|L| \times |A|}$
- A row for each link and a column for each actor

$$\Gamma_{l,a} = \begin{cases} 0 & \text{if } a \text{ is not an endpoint of } l \\ prod(l) & \text{if } a \text{ is the start-point of } l \\ -cons(l) & \text{if } a \text{ is the end-point of } l \end{cases}$$

Topology Matrix

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 0 \\ 0 & 0 & 0 & 3 & 0 & -2 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & -3 \\ 0 & 0 & 0 & 0 & 0 & 8 & -9 \end{bmatrix}$$

Repetitions Vector

- Topology matrix used to schedule actors
- Repetitions Vector is an invocation of actors counter
- if every actor is invoked as many times as the RV says, the number of tokens after the execution(s) is the same as before
  - i.e. $\Gamma \mathbf{r} = \mathbf{0}$
- called "steady state"

$$\begin{bmatrix} 6 \\ 6 \\ 12 \\ 6 \\ 18 \\ 9 \\ 8 \end{bmatrix}$$

Schedules

Pediodic Schedule

A list of actor invocations that return to the steady state

Schedules

Pediodic Schedule

A list of actor invocations that return to the steady state

Init Schedule

A list of actor invocations that flood the buffers to the desired level

Schedules

Pediodic Schedule

A list of actor invocations that return to the steady state

Init Schedule

A list of actor invocations that flood the buffers to the desired level

"Death Schedule"

A list of actor invocations that drain the buffers to the initial level (i.e. with the given delays)

Periodic

- Execution order doesnt affect fill state
- May require prohibitively large buffers
- Cant execute if we dont have data in the buffers
- we can fiddle with the order to reduce buffer size (NP-Hard)

Init and Death

- Fills and clears the buffers
- We have to know in advance how much we want to fill the buffers
- It may be a good idea to fill so that **any** periodic schedule may execute
- presume all actors simultaneously execute all their repetitions at the start
- delay actor invocations if we know buffers wont be full.
  - Strategically place actors
  - avoid costly dynamic scheduling

Cloud Applications

- massive virtual machines/multiple virtual machines
- network bandwidth is an important consideration
- small probability of node failure, whole computation is gone
- develop schemes to ensure fault tolerance
  - Statically replicate multiple nodes
  - Dynamically re-compute if a node fails
- examine trade-off between Makespan, Network Cost, and Fault tolerance

N-Process Machines

- abstraction from (single-core/multi-core/multi-processor/cloud) machines
- Use cost matrices to account for varying bandwith/processing capabilities
  - 2 cores in the same machine have 0 communication cost
  - Actors assigned to a 1ghz processor take longer than those on a 1.8ghz processor
- Use this machine specification to generate a mapping of actors to processors
- Most must be approximated, optimality is NP-Hard

References I

- S. S. Battacharyya, P. K. Murthy, and E. A. Lee.
  Software Synthesis from Dataflow Graphs.
  Kluwer Academic Publishers, 101 Philip Drive
  Assinippi Park
  Norwell, Massachusetts 02061 USA, first Ed., 1996.

- E. A. Lee and D. G. Messerschmitt.
  "Static scheduling of synchronous data flow programs for digital signal processing".
  Computers, IEEE Transactions on, Vol. 100, No. 1, pp. 24–35, 1987.