ESE535: Electronic Design Automation

Day 13: February 27, 2013
Dataflow

Previously

• Scheduling of concurrent operations

Want to See

• Abstract compute model
  – natural for parallelism and hardware

• Describe computation abstracted from implementation
  – Defines correctness

Today

• Dataflow
• SDF
  – Single rate
  – Multirate
• Dynamic Dataflow
• Expression

Parallelism Motivation

Producer-Consumer Parallelism

• Can run concurrently
• Just let consumer know when producer sending data
Pipeline Parallelism

- Can potentially all run in parallel
- Like physical pipeline
- Useful to think about stream of data between operators

DAG Parallelism

- Doesn’t need to be linear pipeline
- Synchronize inputs

Graphs with Feedback

- In general may hold state
- Very natural for many tasks

Definitions

Dataflow / Control Flow

Dataflow
- Program is a graph of operators
- Operator consumes tokens and produces tokens
- All operators run concurrently

Control flow (e.g. C)
- Program is a sequence of operations
- Operator reads inputs and writes outputs into common store
- One operator runs at a time
  - defines successor

Token

- Data value with presence indication
  - May be conceptual
    - Only exist in high-level model
    - Not kept around at runtime
  - Or may be physically represented
    - One bit represents presence/absence of data
Token Examples?

- What are familiar cases where data may come with presence tokens?
  - Network packets
  - Memory references from processor
    - Variable latency depending on cache presence
  - Start bit on serial communication

Operator

- Takes in one or more inputs
- Computes on the inputs
- Produces results

  - Logically self-timed
    - “Fires” only when input set present
    - Signals availability of output

Dataflow Graph

- Represents
  - computation sub-blocks
  - linkage
- Abstractly
  - controlled by data presence

Dataflow Graph Example

In-Class Dataflow Example
Stream

- Logical abstraction of a persistent point-to-point communication link
  - Has a (single) source and sink
  - Carries data presence / flow control
  - Provides in-order (FIFO) delivery of data from source to sink

Streams

- Captures communications structure
  - Explicit producer→consumer link up
- Abstract communications
  - Physical resources or implementation
  - Delay from source to sink

Register Transfer Level (RTL)

- Describe computation as logic and registers
- Equations (logic) define values to be clocked into next register
- Typically what you right in VHDL, Verilog

Dataflow Abstracts Timing

- Doesn’t say
  - on which cycle calculation occurs [contrast RTL]
- Does say
  - What order operations occur in
  - How data interacts
    - i.e. which inputs get mixed together
- Permits
  - Scheduling on different # of resources
  - Operators with variable delay [examples?]
  - Variable delay in interconnect [examples?]

Examples

- Operators with Variable Delay
  - Cached memory or computation
  - Shift-and-add multiply
  - Iterative divide or square-root
- Variable delay interconnect
  - Shared bus
  - Distance changes
    - Wireless, longer/shorter cables
  - Computation placed on different cores?

Difference:

Dataflow Graph/Pipeline

Behavioral (C, MATLAB, …)

Arch. Select Schedule
RTL
FSM assign
Two-level
Multilevel opt.
Covering
Retiming
Gate Netlist Placement Routing
Layout
Masks
**Clock Independent Semantics**

Interconnect
Takes n-clocks

---

**Semantics**

- Need to implement semantics
  - *i.e.* get same result as if computed as indicated
- But can implement any way we want
  - That preserves the semantics
  - Exploit freedom of implementation

---

**Dataflow Variants**

**Synchronous Dataflow (SDF)**

- Particular, restricted form of dataflow
- Each operator
  - Consumes a fixed number of input tokens
  - Produces a fixed number of output tokens
  - When full set of inputs are available
    - Can produce output
    - Can fire any (all) operators with inputs available at any point in time

---

**Synchronous Dataflow**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

**SDF: Execution Semantics**

```plaintext```
while (true)
  Pick up any operator
  If operator has full set of inputs
    Compute operator
    Produce outputs
    Send outputs to consumers
```
Multirate Synchronous Dataflow

- Rates can be different
  - Allow lower frequency operations
  - Communicates rates to CAD
    - Something not clear in RTL
    - Use in scheduling, provisioning
  - Rates must be constant
    - Data independent

SDF

- Can validate flows to check legal
  - Like KCL → token flow must be conserved
  - No node should
    - be starved of tokens
    - Collect tokens
- Schedule onto processing elements
  - Provisioning of operators
  - Provide real-time guarantees
- Simulink is SDF model

SDF: good/bad graphs

Dynamic Rates?

- When might static rates be limiting?
  - Compress/decompress
    - Lossless
    - Even Run-Length-Encoding
  - Filtering
    - Discard all packets from geraldo
  - Anything data dependent

Data Dependence

- Add Two Operators
  - Switch
  - Select
Switch

Filtering Example

Select

Constructing If-Then-Else

Select Example

Looping

• for (i=0;i<Limit;i++)
**Universal**

- Once we add switch and select, the dataflow model is as powerful as any other
  - E.g. can do anything we could do in C
  - “Turing Complete” in formal CS terms

**Dynamic Challenges**

- In general, cannot say
  - If a graph is well formed
    - Will not deadlock
  - How many tokens may have to buffer in stream
  - Right proportion of operators for computation

**Expression**

- Could express operators in C/Java
  - Each is own thread
- Link together with Streams
- E.g. SystemC

**C Example**

```c
while (!is eof(stream_a) && !is eof(stream_b))
    A = stream_a.read();
    B = stream_b.read();
    Out = (a + b) * (a - b);
    stream_out.write(Out);
```

**Connecting up Dataflow**

```c
stream stream1 = new stream();
operator prod = new stock(stream1);
operator cons = new encrypt(stream1);
```
What’s the Point?

• Seen repeatedly: exploit freedom
  – To reduce costs
• A1: How do we capture freedom that exists in computational?
  – Higher-level than an implementation
  – Perhaps as a useful intermediate
• A2: How do we allow freedom for implementations (or instances) to take variable time?

Summary

• Dataflow Models
  – Simple pipelines
  – DAGs
  – SDF (single, multi)-rate
  – Dynamic Dataflow
• Allow
  – express parallelism
  – freedom of implementation

Big Ideas:

• Dataflow
  – Natural model for capturing computations
  – Communicates useful information for optimization
    • Linkage, operator usage rates
• Abstract representations
  – Leave freedom to implementation

Admin

• Homework 4 Due Today
• Spring Break next week
• Back on Monday 3/11
  – Reading on Blackboard