Lecture 4
Data-Flow Scheduling

Forrest Brewer
Data Flow Model Hierarchy

- Kahn Process Networks (KPN) (asynchronous task network)
- Dataflow Networks
  - special case of KPN
  - actors, tokens and firings
- Static Data Flow (Clocked Automata)
  - special case of DN
  - static scheduling
  - code generation
  - buffer sizing (resources!!)
- Other Clocked Data Flow models
  - Boolean Data Flow
  - Dynamic Data Flow
  - Sequence Graphs, Dependency Graphs, Data Flow Graphs
  - Control Data Flow
Data Flow Models

- Powerful formalism for data-dominated system specification
- Partially-ordered model (over-specification)
- Deterministic execution independent of scheduling

Used for
- simulation
- scheduling
- memory allocation
- code generation

for Digital Signal Processors (HW and SW)
Data Flow Networks

- A Data Flow Network is a collection of actors which are connected and communicate over unbounded FIFO queues.
- Actors firing follows firing rules:
  - Firing rule: number of required tokens on inputs
  - Function: number of consumed and produced tokens
- Actors are functional i.e. have no internal state
- Breaking processes of KPNs down into smaller units of computation makes implementation easier (scheduling)
- Tokens carry values:
  - integer, float, audio samples, image of pixels
- Network state: number of tokens in FIFOs
Intuitive semantics

- At each time, one actor is fired
  - Can fire more – but one is always safe (atomic firing)
- When firing, actors consume input tokens and produce output tokens
- Actors can be fired only if there are enough tokens in the input queues
Filter example

- **Example: FIR filter**
  - single input sequence $i(n)$
  - single output sequence $o(n)$
  - $o(n) = c_1 i(n) + c_2 i(n-1)$
Filter example

- Example: FIR filter
  - single input sequence \(i(n)\)
  - single output sequence \(o(n)\)
  - \(o(n) = c_1 \cdot i(n) + c_2 \cdot i(n-1)\)
Filter example

- Example: FIR filter
  - single input sequence i(n)
  - single output sequence o(n)
  - o(n) = c1 i(n) + c2 i(n-1)
Filter example

- Example: FIR filter
  - single input sequence $i(n)$
  - single output sequence $o(n)$
  - $o(n) = c1 \ i(n) + c2 \ i(n-1)$
Filter example

- **Example: FIR filter**
  - single input sequence \( i(n) \)
  - single output sequence \( o(n) \)
  - \( o(n) = c1 \, i(n) + c2 \, i(n-1) \)
Example: FIR filter
- single input sequence \( i(n) \)
- single output sequence \( o(n) \)
- \( o(n) = c_1 i(n) + c_2 i(n-1) \)
Filter example

- Example: FIR filter
  - single input sequence \( i(n) \)
  - single output sequence \( o(n) \)
  - \( o(n) = c_1 \, i(n) + c_2 \, i(n-1) \)
Filter example

- Example: FIR filter
  - single input sequence \( i(n) \)
  - single output sequence \( o(n) \)
  - \( o(n) = c1 \ i(n) + c2 \ i(n-1) \)
Filter example

- Example: FIR filter
  - single input sequence $i(n)$
  - single output sequence $o(n)$
  - $o(n) = c_1 i(n) + c_2 i(n-1)$
Example: FIR filter
- single input sequence \( i(n) \)
- single output sequence \( o(n) \)
- \( o(n) = c1 \cdot i(n) + c2 \cdot i(n-1) \)
Filter example

- **Example: FIR filter**
  - single input sequence $i(n)$
  - single output sequence $o(n)$
  - $o(n) = c_1 i(n) + c_2 i(n-1)$
Examples of Data Flow actors

- **SDF: Synchronous (or Static) Data Flow**
  - fixed number of input and output tokens per invocation

- **BDF: Boolean Data Flow**
  - control token determines consumed and produced tokens
Examples of Data Flow actors

- **Sequence Graphs, Dependency Graph, Data Flow Graph**
  - Each edge corresponds to exactly one value
  - No buffering
  - Special Case of SDF

- **CDFG: Control Data Flow Graphs**
  - Adds branching (conditionals) and iteration constructs
  - Many different models for this

Typical model in many behavioral/architectural synthesis tools
Scheduling Data Flow

- Given a set of Actors and Dependencies
- How to construct valid execution sequences?
  - Static Scheduling:
    Assume that you can predefine the execution sequence
  - FSM Scheduling:
    Sequencing defined as control-dependent FSM
  - Dynamic Scheduling
    Sequencing determined dynamically (run-time) by predefined rules
- In all cases, need to not violate resource or dependency constraints
- In general, both actors and resources can themselves have sequential (FSM) behaviors
(MIPS) RISC Instruction Execution

Operand Dependencies

Tasks

pc in

incr pc

fetch inst

inst out

pc out

inst in

read rs

read rt

alu

write rd
Another RISC Instruction Execution

Operand Resolution
Complete Instruction Task Graph

PC in

Incr PC

Adjust PC

Fetch Inst

Inst Out

Inst Out

PC Out

Inst In

Inst Decode

Read RS

Read RT

Alu

Write Mem

Read Mem

Branch Compare

Write RD
Hierarchical Data Flow

- Low Level Task graphs can be composed into higher level graphs
  - Internal State
  - Side Effects
- Promote interior I/O interfaces to act for higher level blocks
- Note PC update and increment to support concurrency with data-path
Scheduling Result: Valid Sequences

RISC Instruction Task
Scheduling Result: Valid Sequences

RISC Instruction Task
Scheduling Result: Valid Sequences

RISC Instruction Task

3
Scheduling Result: Valid Sequences

RISC Instruction Task

inst out

pc out

4
Operation (unit) Scheduling

- On the way to task scheduling, a very important case is that where there is no storage on the edges and the duration of the actors is a multiple of some clock
  - No fifo implies that each value is transient and will be lost if not captured by the next operator
  - Imposition of a clock allows use of RT-level modeling (e.g. Verilog or VHDL)
    Create a register for each data edge that crosses a clock boundary
- This model is useful for Compiler Level data-flow as well as RT-level modeling
Synthesis in Temporal Domain

- Scheduling and binding can be done in different orders or together
- Schedule:
  - Mapping of operations to time slots + binding to resources
  - A scheduled sequencing graph is a labeled graph

[©Gupta]
Operation Types

- Operations have *types*
- Each resource may have several types and timing constraints
- $T$ is a relation that maps an operation to a resource by matching types
  - $T : V \rightarrow \{1, 2, \ldots, \text{nres}\}$.
- In general:
  - A resource type may implement more than one operation type (ALU)
  - May have family of timing constraints (data-dependent timing?!) 
- Resource binding:
  - Notion of exclusive mapping
    - Pipeline resources or other state?
    - Arbitration
  - Choice linked to complexity of interconnect network
Schedule in Spatial Domain

- Resource sharing
  - More than one operation bound to same resource
  - Operations serialized
  - Can be represented using hyperedges (Graph Vertex Partition)

[Diagram of a graph with nodes labeled as NOP, 1, 2, 3, 4, and operations indicated by symbols such as + and -]
Scheduling and Binding

- **Resource constraints:**
  - Number of resource instances of each type \{ak : k=1, 2, ..., nres\}.
  - Link, register, and communication resources

- **Scheduling:**
  - Timing of operation

- **Binding:**
  - Location of operation

- **Costs:**
  - Resources \(\approx\) area (power?)
  - Registers, steering logic (Muxes, busses), wiring, control unit

- **Metric:**
  - Start time of the “sink” node
  - Might be affected by steering logic and schedule (control logic) – resource-dominated vs. ctrl-dominated
Architectural Optimization

- Optimization in view of design space flexibility
- A multi-criteria optimization problem:
  - Determine schedule $f$ and binding $b$.
  - Given area $A$, latency $l$ and cycle time $t$ objectives
- Find non-dominated points in solution space
  - Pareto-optimal solutions
- Solution space tradeoff curves:
  - Non-linear, discontinuous
  - Area / latency / cycle time (Power?, Slack?, Registers?, Simplicity?)
- Evaluate (estimate) cost functions
- Constrained optimization problems for resource dominated circuits:
  - Min area: solve for minimal binding
  - Min latency: solve for minimum $l$ scheduling
Operation Scheduling

- **Input:**
  - Sequencing graph $G(V, E)$, with $n$ vertices
  - Cycle time $t$
  - Operation delays $D = \{d_i: i=0..n\}$

- **Output:**
  - Schedule $f$ determines start time $t_i$ of operation $v_i$
  - Latency $l = t_n - t_0$

- **Goal:** determine area / latency tradeoff

- **Classes:**
  - Unconstrained
  - Latency or Resource constrained
  - Hierarchical (accommodate control transfer!)
  - Loop/Loop Pipelined

[©Gupta]
Min Latency Unconstrained Scheduling

- Simplest case: no constraints, find min latency
- Given set of vertices $V$, delays $D$ and a partial order $>$ on operations $E$, find an integer labeling of operations $\phi: V \to \mathbb{Z}^+$ Such that:
  - $t_i = \phi(v_i)$.
  - $t_i \geq t_j + d_j \quad \forall (v_j, v_i) \in E$.
  - $\lambda = t_n - t_0$ is minimum.
- Solvable in polynomial time
- Bounds on latency for resource constrained problems

Algorithm? ASAP algorithm used: topological order
ASAP Schedules

- Schedule $v_0$ at $t_0=0$.
- While ($v_n$ not scheduled)
  - Select $v_i$ with all scheduled predecessors
  - Schedule $v_i$ at $t_i = \max \{t_j + d_j\}$, $v_j$ being a predecessor of $v_i$.
- Return $t_n$.
ALAP Schedules

- Schedule $v_n$ at $t_0 = \lambda$.
- While ($v_0$ not scheduled)
  - Select $v_i$ with all scheduled successors
  - Schedule $v_i$ at $t_i = \min \{t_j - d_j\}$, $v_j$ being a successor of $v_i$. 

![Diagram of ALAP Schedules]
Resource Constraint Scheduling

- Constrained scheduling
  - General case NP-complete (3 or more resources)
  - Minimize latency given constraints on area or the resources (ML-RCS)
  - Minimize resources subject to bound on latency (MR-LCS)

- Exact solution methods
  - ILP: Integer Linear Programming (Lin, Gebotys)
  - Symbolic Scheduling (Haynal, Radevojevic)
  - Hu’s heuristic algorithm for identical processors

- Heuristics
  - List scheduling
  - Force-directed scheduling
  - Taboo search, Monte-Carlo, many others…
Linear Programming

- A linear program consists of a set of real variables, a set of linear constraints on the variables and a linear objective function
  - A set of *feasible points*, each characterized by a vector of real values satisfying all the linear constraints may exist.
  - Because each linear constraint describes a half-space, with points on one side being feasible, the intersection of the half spaces, if it exists is a *convex hull*.
  - The objective function can be characterized as a set of level planes with the objective increasing along a vector normal to the planes.
  - Since the feasible points are convex, a maximal feasible point occurs one or more hull vertices.
Why Linear Programming?

- Linear programs provide a simple way to express a large variety of optimization problems
  - Complexity is polynomial for real variables, NP-hard for integer, binary or mixed constraints

Consider a simple knapsack problem: fill a sack with objects from a list, whose total weight does not exceed a limit M, but is at large as possible.

Let $x_i$ be a binary (0 or 1) variable, if the weight of object $i$ is $c_i$, the knapsack total weight is:

$$w = \sum_{i=0}^{n} x_i c_i$$
Simplified ILP Formulation

- Use binary decision variables
  - \( i = 0, 1, \ldots, n \)
  - \( l = 1, 2, \ldots, \lambda + 1 \) \( \lambda ' \) given upper-bound on latency
  - \( x_{il} = 1 \) if operation \( i \) starts at step \( l \), 0 otherwise.

- Set of linear inequalities (constraints), and an objective function (min latency)

- Observations:
  - \( x_{il} = 0 \) for \( l < t_{i}^{S} \) and \( l > t_{i}^{L} \)
  - \( t_{i}^{S} = ASAP(v_{i}), t_{i}^{L} = ALAP(v_{i}) \)
  - \( t_{i} = \sum_{l} l \cdot x_{il} \) \( t_{i} = \) start time of op \( i \).
  - \( \sum_{m=l-d_{i}+1}^{l} x_{im} = 1 \) \( \Rightarrow \) is op \( v_{i} \) (still) executing at step \( l \)?
Start Time vs. Execution Time

- Each operation $v_i$, exactly one start time
- If $d_i = 1$, then the following questions are the same:
  - Does operation $v_i$ start at step $l$?
  - Is operation $v_i$ running at step $l$?
- But if $d_i > 1$, then the two questions should be formulated as:
  - Does operation $v_i$ start at step $l$?
    - Does $x_{il} = 1$ hold?
  - Is operation $v_i$ running at step $l$?
    - Does the following hold?

$$\sum_{m=l-d_i+1}^{l} x_{im} = 1$$
Operation $\nu_i$ Still Running at Step $l$?

- Is $\nu_9$ running at step 6?
  - Is $x_{9,6} + x_{9,5} + x_{9,4} = 1$?

- Note:
  - Only one (if any) of the above three cases can happen
  - To meet resource constraints, we have to ask the same question for ALL steps, and ALL operations of that type
ILP Formulation of ML-RCS (cont.)

- **Constraints:**
  - Unique start times: \( \sum_l x_{il} = 1, \quad i = 0, 1, \ldots, n \)
  - Sequencing (dependency) relations must be satisfied:
    \[
    t_i \geq t_j + d_j \quad \forall (v_j, v_i) \in E \iff \sum_l l \cdot x_{il} \geq \sum_l l \cdot x_{jl} + d_j
    \]
  - Resource constraints:
    \[
    \sum_{i: T(v_i) = k} \sum_{m = l - d_i + 1}^{l} x_{im} \leq a_k, \quad k = 1, \ldots, n_{res}, \quad l = 1, \ldots, \bar{\lambda} + 1
    \]
- **Objective:** \( \min c^T t \).
  - \( t \) = start times vector, \( c \) = cost weight (e.g., \([0 \ 0 \ \ldots \ 1]\))
  - When \( c = [0 \ 0 \ \ldots \ 1] \), \( c^T t = \sum_l l \cdot x_{nl} \)
First, perform ASAP and ALAP (\( \lambda = 4 \))
– (we can write the ILP without ASAP and ALAP, but using ASAP and ALAP will simplify the inequalities)
ILP Example: Unique Start Times Constraint

Without using ASAP and ALAP values:

\[ x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1 \]
\[ x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1 \]
\[ \ldots \]
\[ x_{11,1} + x_{11,2} + x_{11,3} + x_{11,4} = 1 \]

Using ASAP and ALAP:

\[ x_{1,1} = 1 \]
\[ x_{2,1} = 1 \]
\[ x_{3,2} = 1 \]
\[ x_{4,3} = 1 \]
\[ x_{5,4} = 1 \]
\[ x_{6,1} + x_{6,2} = 1 \]
\[ x_{7,2} + x_{7,3} = 1 \]
\[ x_{8,1} + x_{8,2} + x_{8,3} = 1 \]
\[ x_{9,2} + x_{9,3} + x_{9,4} = 1 \]
\[ \ldots \]
ILP Example: Dependency Constraints

- Using ASAP and ALAP, the non-trivial inequalities are: (assuming unit delay for + and *)

\[
\begin{align*}
2 \cdot x_{7,2} + 3 \cdot x_{7,3} - x_{6,1} - 2 \cdot x_{6,2} - 1 & \geq 0 \\
2 \cdot x_{9,2} + 3 \cdot x_{9,3} + 4 \cdot x_{9,4} - x_{8,1} - 2 \cdot x_{8,2} - 3 \cdot x_{8,3} - 1 & \geq 0 \\
2 \cdot x_{11,2} + 3 \cdot x_{11,3} + 4 \cdot x_{11,4} - x_{10,1} - 2 \cdot x_{10,2} - 3 \cdot x_{10,3} - 1 & \geq 0 \\
4 \cdot x_{5,4} - 2 \cdot x_{7,2} - 3 \cdot x_{7,3} & \geq 0 \\
5 \cdot x_{n,5} - 2 \cdot x_{9,2} - 3 \cdot x_{9,3} - 4 \cdot x_{9,4} - 1 & \geq 0 \\
5 \cdot x_{n,5} - 2 \cdot x_{11,2} - 3 \cdot x_{11,3} - 4 \cdot x_{11,4} - 1 & \geq 0
\end{align*}
\]
ILP Example: Resource Constraints

- Resource constraints (assuming 2 adders and 2 multipliers)

\[
\begin{align*}
    x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} & \leq 2 \\
    x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} & \leq 2 \\
    x_{7,3} + x_{8,3} & \leq 2 \\
    x_{10,1} & \leq 2 \\
    x_{9,2} + x_{10,2} + x_{11,2} & \leq 2 \\
    x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} & \leq 2 \\
    x_{5,4} + x_{9,4} + x_{11,4} & \leq 2 
\end{align*}
\]

- Objective: Min \( X_{n,4} \)
ILP Formulation of Resource Minimization

- Dual problem to Latency Minimization
- Objective:
  - Goal is to optimize total resource usage, \( a \).
  - Objective function is \( c^T a \), where entries in \( c \) are respective area costs of resources.
- Constraints:
  - Same as ML-RCS constraints, plus:
  - Latency constraint added:
    \[
    \sum_l l \cdot x_{nl} \leq \bar{\lambda} + 1
    \]
  - Note: unknown \( a_k \) appears in constraints.
Hu’s Algorithm

- Simple case of the scheduling problem
  - All operations have unit delay
  - All operations (and resources) of the same type
  - Graph is forest

- Hu’s algorithm
  - Greedy
  - Polynomial AND optimal
  - Computes lower bound on number of resources for a given latency
    OR: computes lower bound on latency subject to resource constraints
Basic Idea: Hu’s Algorithm

- Relies on labeling of operations
  - Based on their distances from the sink
  - Length of the longest path passing through that node
- Try to schedule nodes with higher labels first (i.e., most “critical” operations have priority)
- Schedule $a$ nodes at a time
  - $a$ is the number of resources
  - Only schedule nodes that have all their parent/predecessor’s scheduled
  - Each time you schedule one time step (start with step 1, 2, 3, …)
Hu’s Algorithm:

HU (G(V,E), a) {
    Label the vertices // label = length of longest path
        passing through the vertex
    \[ l = 1 \]
    repeat {
        U = unscheduled vertices in V whose
            predecessors have been scheduled
            (or have no predecessors)
        Select S \subseteq U such that \(|S| \leq a\) and labels in S
            are maximal
        Schedule the S operations at step \(l\) by setting
            \(t_i = l, i: v_i \in S.\)
            \(l = l + 1\}
        until \(v_n\) is scheduled. }
Hu’s Algorithm: Example

Step 1: Label Vertices (Assume all operations have unit delays):
Hu’s Algorithm: Example

Find unscheduled vertices with scheduled parents; pick 3 (num. resources) that maximize labels
Hu’s Algorithm: Example

Repeat until all nodes are scheduled
List Scheduling

- Heuristic methods for RCS and LCS
  - Does NOT guarantee optimum solution
- Similar to Hu’s algorithm
  - Greedy strategy
  - Operation selection decided by criticality
  - O(n) time complexity
- More general input
  - Works on general graphs (unlike Hu’s)
  - Resource constraints on different resource types
List Scheduling Algorithm: ML-RCS

LIST_L (G(V,E), a) {
    l = 1
    repeat {
        repeat {
            for each resource type k {
                U_{l,k} = available vertices in V
                T_{l,k} = operations in progress.
                Select S_k \subseteq U_{l,k} such that |S_k| + |T_{l,k}| \leq a_k
                Schedule the S_k operations at step l
            }
        }
        l = l + 1
    } until v_n is scheduled.
}
List Scheduling Example

Assumptions: three multipliers with latency 2; 1 ALU with latency 1
List Scheduling Algorithm: MR-LCS

\[
\text{LIST}_R \left( G(V,E), \lambda' \right) \{ \\
\quad a = 1, \quad l = 1 \\
\quad \text{Compute the ALAP times } t^L. \\
\quad \text{if } t_0^L < 0 \\
\quad \quad \text{return (not feasible)} \\
\quad \text{repeat} \\
\quad \quad \text{for each resource type } k \{ \\
\quad \quad \quad U_{l,k} = \text{available vertices in } V. \\
\quad \quad \quad \text{Compute the slacks } \{ s_i = t_i^L - l, \forall v_i \in U_{l,k} \}. \\
\quad \quad \quad \text{Schedule operations with zero slack, update } a \\
\quad \quad \quad \text{Schedule additional } S_k \subseteq U_{l,k} \text{ under } a \text{ constraints} \\
\quad \quad \} \\
\quad l = l + 1 \\
\quad \text{until } v_n \text{ is scheduled.} 
\}
Control Dependency in Scheduling

- Practical Programs often have behavior that is dependant on a few conditions. Such conditions are called “control” variables and are usually Boolean or short Enumerations.
  - Effects incorporated in Data-Flow by making the dependencies multi-valued, with selection by the dynamic value of some control variable
  - Program controls can be modeled by marking every dependency entering or leaving a basic block, using scope and sequencing rules to identify dependent targets

- Issue: Controls nest making the number of dependent paths grow exponentially fast
  - How to avoid blow-up of the problem representation?
A Scheduled CDFG

- **Cycle 0**
- **Cycle 1**
- **Cycle 2**

- Speculated Operation
- Two-Cycle Function Unit
- Resource Bounds and Causality Satisfied
Operations as One-Bit NFAs

- $0 \rightarrow 0$ Operation unscheduled and remains so
- $0 \rightarrow 1$ Operation scheduled next cycle
- $1 \rightarrow 1$ Operation scheduled and remains so
- $1 \rightarrow 0$ Operation scheduled but result lost
Product of all One-Bit NFAs form Scheduling NFA

- Compressed ROBDD representation
- State represents subset of completed operations
- Constraints modify transition relation
Resource Bounds

- Operation’s $0 \rightarrow 1$ indicates resource use
- Resource bounds limit simultaneous $0 \rightarrow 1$ in scheduling NFA

- ROBDD representation:
  - operations choose bound
  - $2 \times bound \times operations$ nodes
  - easy to build & compressed
Dependency Implication

$A =$ “Operation $j$ is scheduled”

$B =$ “All of operation $j$’s predecessors are known”

False implication cubes $(A\overline{B})$ removed from transition relation

$$\sum_{i \rightarrow j} \overline{P}_i N_j$$ where $i \rightarrow j$ is a dependency arc in the CDFG
Valid DFG Scheduling NFA

- Example DFG, 1 resource
- NFA transition relation implicitly represents graph
- Any path from all operations unknown to all known is a valid schedule
- Shortest path is minimum latency schedule
All Minimum Latency Schedules

- Symbolic reachable state analysis
  - Newly reached states are saved each cycle
  - Backward pruning preserves transitions used in all shortest paths
All Minimum Latency Schedules

- Symbolic reachable state analysis
  - Newly reached states are saved each cycle
  - Backward pruning preserves transitions used in all shortest paths
All Minimum Latency Schedules

- Symbolic reachable state analysis
  - Newly reached states are saved each cycle
  - Backward pruning preserves transitions used in all shortest paths
All Minimum Latency Schedules

- Symbolic reachable state analysis
  - Newly reached states are saved each cycle
  - Backward pruning preserves transitions used in all shortest paths
All Minimum Latency Schedules

- Symbolic reachable state analysis

- Not bound to reachable state analysis
  - Refinements or heuristics to find subset of shortest paths
  - Other objectives besides shortest paths
CDFGs: Multiple Control Paths

- Guard automata differentiate control paths
  - Before control operation scheduled:
    - Control may change value, unknown
  - After control operation scheduled:
    - Control value fixed, known
CDFGs: Multiple Control Paths

- All control paths form ensemble schedule
  - Possibly 2c control paths to schedule
- Added dummy operation identifies when a control path terminates
  - Only one termination operation, not 2c
- Ensemble schedule may not be causal
  - Solution: validation algorithm
Join Dependency Implication

\[ A = \text{“Operation } j \text{ is scheduled”} \]

\[ B = \text{“The join control resolution is known and all of operation } j \text{’s resolved predecessors are known”} \]
Operation Exclusion

- Control unknown
  - Speculation possible
  - Resource bounds applicable to both branches

- Control known
  - Branch resource bounds mutually exclusive
  - Other branch’s operations not required
CDFG Example

- One green resource
- Shortest paths
- False termination
Validated CDFG Example

- Validation algorithm ensures control paths don’t bifurcate before control value is known
Validated CDFG Example

- Validation algorithm ensures control paths don’t bifurcate before control value is known
- Pruned for all shortest paths as before
Automata-based Scheduling Conclusions

- Efficient encoding
  - No pruning used!
  - Breadth-first search consolidates schedules with common histories

- All valid schedules found
  - Further refinements and heuristics possible

- Despite exact nature, representation growth is minimized
  - $O(<\text{operations}>*<\text{cycles}>*<\text{controls}>)$
Construction for Looping DFG’s

- Use trick: 0/1 representation of the MA could be interpreted as 2 mutually exclusive operand productions
- Schedule from ~know -> known -> ~known where each 0->1 or 1->0 transition requires a resource.
- Since dependencies are on operands, add new dependencies in 1 ->0 sense as well
- Idea is to remove all transitions which do not have complete set of known or ~known predecessors for respective sense of operation

- So -- get looping DFG automata as nearly same automata as before
  - preserve efficient representation
- Selection of “Minimal Latency” solutions is more difficult
Loop construction: resources

- Resources: we now count both 0 -> 1 and 1 -> 0 transition as requiring a resource.
- Use “Tuple” BDD construction: at most k bits of n BDD
- Despite exponential number of product terms, BDD complexity: O(bound * |V|)
Example CA

- State order \((v_1, v_2, v_3, v_4)\)

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>0,1,8,9</td>
</tr>
<tr>
<td>2,3</td>
<td>0,1,2,3,8,9,A,B</td>
</tr>
<tr>
<td>4,5,C,D</td>
<td>4,5,6,7,C,D,E,F</td>
</tr>
<tr>
<td>6,7,A,B</td>
<td>2,3,6,7,A,B,E,F</td>
</tr>
<tr>
<td>8,9</td>
<td>0,1,4,5,8,9,C,D</td>
</tr>
<tr>
<td>E,F</td>
<td>6,7,E,F</td>
</tr>
</tbody>
</table>

- Path \(0,9,C,7,2,9,C,7,2,\ldots\) is a valid schedule.
- By construction, only 1 instance of any operator can occur in a state.
Strategy to Find Maximal Throughput

- CA automata construction simple
- How to find closed subset of paths guaranteeing optimal throughput
- Could start from known initial state and prune slow paths as before-- but this is not optimal!

- Instead: find all reachable states (without resource bounds)
- Use state set to prune unreachable transitions from CA
- Choose operator at random to be pinned (marked)
- Propagate all states with chosen operator until it appears again in same sense
- Verify closure of constructed paths by Fixed Point iteration
- If set is empty -- add one clock to latency and verify again

- Result is maximal closed set of paths for which optimal throughput is guaranteed
Maximal Throughput Example

- DFG above has closed 3-cycle solution (2 resources)
- However- average latency is 2.5-cycles
- (a,d) (b,e) (a,c) (b,d) (c,e) (a,d) ...
- Requires 5 states to implement optimal throughput instance
- In general, it is possible that a k-cycle closed solution may exist, even if no k-state solution can be found
- Current implementation finds all possible k-cycle solutions
Schedule Exploration: Loops

- **Idea:** Use partial symbolic traversal to find states bounding minimal latency paths
- **Latency:** Identify all paths completing cycle in given number of steps
- **Repeatability:** Fixed Point Algorithm to eliminate all paths which cannot repeat in given latency
- **Validation:** Ensure all possible control paths are present for each remaining path
- **Optimization:** Selection of Performance Objective
Kernel Execution Sequence Set

- Path from Loop cut to first repeating states
- Represents candidates for loop kernel
Repeatable Kernel Execution Sequence Set

- Fixed-point prunes non-repeating states
  - Only repeatable loop kernels remain
  - Paths not all same length
  - Average latency <= shortest Repeating Kernel

---

Repeatable Loop Kernel

$\text{Loop Cut}$
Validation I

- Schedule Consists of bundle of compatible paths for each possible future
- Not Feasible to identify all schedules
- Instead, eliminate all states which do not belong to some ensemble schedule
- Fragile since any further pruning requires re-validation
- Double fixed point
Validation II

- Path Divergence -- Control Behavior
  - Ensure each path is part of some complete set for each control outcome
  - Ensure that each set is Causal
Loop Cuts and Kernels

- Method Covers all Conventional Loop Transformations
  - Sequential Loop
  - Loop winding
  - Loop Pielining
Results

- Conventional Scheduling
  - 100-500x speedup over ILP
- Control Scheduling: Complexity typically pseudo polynomial in number of branching variables
- Cyclic Scheduling:
  - Reduced preamble complexity
  - Capacity: 200-500 operands in exact implementation
- General Control Dominated Scheduling:
  - Implicit formulation of all forms of CDFG transformation
  - Exact Solutions with Millions of Control paths
- Protocol Constrained Scheduling:
  - Exact for small instances – needs sensible pruning of domain
Conclusions

- ILP – optimal, but exponential runtime (often)
- Hu’s
  - Optimal and polynomial
  - Very restricted cases
- List scheduling
  - Extension to Hu’s for general case
  - Greedy (fast) \( O(n^2) \) but suboptimal
- Automata-Based Scheduling
  - Manages controls and some speculation
  - Exact, practical to few hundred operations
- Next Time: Task and Process Scheduling