ECE 6775
High-Level Digital Design Automation
Fall 2023

More Scheduling
Announcements

- Lab 1 graded
  - Only 2 integer bits needed for the fixed-point design

- Lab 2 due tomorrow

- Lab 3 will be released soon

- Virtual lecture next Tuesday
Agenda

- ILP for time-constrained scheduling
- Heuristic algorithms for constrained scheduling
  - List scheduling
  - SDC-based scheduling
Exercise: Formulating ILP

- Minimize the number of classrooms that the school must allocate for the following courses
- Steps to formulate the ILP
  1. Create variables
  2. Each course to be scheduled to exactly one of the preferred slots
  3. Determine the number of rooms required (by creating new derived variables)
  4. Set up the objective function

<table>
<thead>
<tr>
<th>Course</th>
<th>Preferred Slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(1) (2)</td>
</tr>
<tr>
<td>B</td>
<td>(1) (3)</td>
</tr>
<tr>
<td>C</td>
<td>(2) (3)</td>
</tr>
<tr>
<td>D</td>
<td>(2)</td>
</tr>
</tbody>
</table>

(1) 8:00 – 10:00am
(2) 10:00am – 12:00pm
(3) 12:00 – 2:00pm

1. \( x_{i,s} \): course i uses slot s

2. \( x_{A,1} + x_{A,2} = 1 \)
   \( x_{B,1} + x_{B,3} = 1 \)
   \( x_{C,2} + x_{C,3} = 1 \)
   \( x_{D,2} = 1 \)

3. \( r_1 = x_{A,1} + x_{B,1} \)
   \( r_2 = x_{A,2} + x_{C,2} + x_{D,2} \)
   \( r_3 = x_{B,3} + x_{C,3} \)

4. Objective: min \( \max \{ r_1, r_2, r_3 \} \)

Linearize

\[
\begin{align*}
\min R \\
R &\geq r_1, R \geq r_2, R \geq r_3
\end{align*}
\]
Time-Constrained Scheduling (TCS)

- Dual problem of resource-constrained scheduling
  - Overall latency is given as a constraint (deadline)
  - Minimize the total cost in terms of area (or resource usage), power, etc.

- NP-hard problem
  - ILP formulation is exact but is not a polynomial-time solution
  - Force-directed scheduling is a well-known heuristic for TCS (see De Micheli chapter 5.4.4)
Example: ILP Formulation for TCS

- ILP for time-constrained scheduling

\[
\text{minimize } c^Ty \\
\begin{align*}
x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} & \leq y_1 \\
x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} & \leq y_1 \\
x_{7,3} + x_{8,3} & \leq y_1 \\
x_{5,4} + x_{9,4} + x_{11,4} & \leq y_2 \\
\end{align*}
\]

\[\ldots\]

What does the \( y \) vector represent?
Recap: Constrained Scheduling in HLS

- Constrained scheduling
  - General case NP-hard
  - Resource-constrained scheduling (RCS)
    - Minimize latency given constraints on area or resources
  - Time-constrained scheduling (TCS)
    - Minimize resources subject to bound on latency

- Exact methods
  - Integer linear programming (ILP)
  - Hu’s algorithm for a very restricted problem

- Heuristics
  - List scheduling
  - Force-directed list scheduling
  - SDC-based scheduling
  ...
List Scheduling

- A widely-used heuristic algorithm for RCS
  - Schedule one control step (cycle) at a time
  - Maintain a list of “ready” operations considering dependence
  - Assign priorities to operations; most “critical” operations (with the highest priorities) go first

- Often refers to a family of algorithms
  - Typically classified by the way priority function is calculated
    • Static priority: Priorities are calculated once before scheduling
    • Dynamic priority calculation: Priorities are updated during scheduling
Static Priority Example: Node Height

Nodes are labelled with distance to sink (height)

Ready operations are colored in green

Assumptions:
- All operations have unit delay
- 2 MULTs, 1 AddSub, and 1 CMP available
Ready Nodes with Highest Priorities Picked First

Assumptions:
- All operations have unit delay
- 2 MULTs, 1 AddSub, and 1 CMP available
Update Ready Nodes and Repeat for Each Step

Assumptions:
- All operations have unit delay
- 2 MULTs, 1 AddSub, and 1 CMP available
Update Ready Nodes and Repeat for Each Step

Assumptions:
- All operations have unit delay
- 2 MULTs, 1 AddSub, and 1 CMP available
Repeat Until All Nodes Scheduled

- Assumptions:
  - All operations have unit delay
  - 2 MULTs, 1 AddSub, and 1 CMP available
A Special Case

▸ With the following (very) restrictive conditions:
  – All operations have unit delay (i.e., single cycle)
  – All operations (and resources) are of the same type
  – Graph is a forest

▸ List scheduling with static height-based priorities guarantees optimality

▸ This is known as Hu’s algorithm
  – Guarantees
HLS Scheduling:
Tension between Scalability and Quality

- **High scalability (w/ greedy decisions)**
  - List scheduling (e.g., [Parker et al., DAC’86])

- **Low quality**
  - Force-directed scheduling
    - [Paulin & Knight, TCAD’89]

- **High quality (w/ global optimization)**
  - Meta heuristics
    - (e.g., Ant colony [Wang et al., TCAD’07])
  - ILP

- **Slow runtime**
  - Handle rich constraints
  - Perform global optimization
  - Archive fast runtime
More Realistic Scheduling Problems

- Operation chaining
  - More compact schedule
- Multi-cycle operations
  - Nonpipelined or pipelined
  - Higher frequency
- Mutually exclusive operations
  - Scheduled in the same step, but with mutually exclusive execution conditions
  - Higher resource utilization
- Other timing constraints
  - Frequency constraints, latency constraints, relative time constraints
A Simple Operation Chaining Problem

**Given:** A chain of $n$ operations. Without any registers, the cycle time equals the total combinational delay, which is $D = \text{sum}(d_i)$.

![Diagram of a chain of operations]

**Question:** How to place TWO registers on the chain to achieve the minimum cycle time?

**Example:**

```
5 → 1 □ → 6 → 2 □ → 7
```
SDC-Based Scheduling

- SDC = System of difference constraints

- Target cycle time: 5ns
- Delay estimates
  - Mul (x): 3ns
  - Add (+): 1ns
  - Load/Store (ld/st): 1ns

\[ s_i : \text{schedule variable for operation } i \]

- Dependence constraints
  - \( <v_0, v_4> : s_0 - s_4 \leq 0 \)
  - \( <v_1, v_3> : s_1 - s_3 \leq 0 \)
  - \( <v_2, v_3> : s_2 - s_3 \leq 0 \)
  - \( <v_3, v_4> : s_3 - s_4 \leq 0 \)
  - \( <v_4, v_5> : s_4 - s_5 \leq 0 \)

- Cycle time constraints
  - \( v_1 \rightarrow v_5 : s_1 - s_5 \leq -1 \)
  - \( v_2 \rightarrow v_5 : s_2 - s_5 \leq -1 \)

Operation chaining is naturally supported.

Timing constraints:

To meet the cycle time, \( v_2 \) and \( v_5 \) should have a minimum separation of one cycle.

[J. Cong & Z. Zhang, DAC, 2006] [Z. Zhang & B. Liu, ICCAD, 2013]
Exercise: Latency Constraint in SDC

How to enforce that operations \( v_3 \) and \( v_4 \) are not chained and at most two cycles apart?

\[
\begin{align*}
&v_0 \xrightarrow{\text{ld}} v_1 \xrightarrow{\text{ld}} v_2 \xrightarrow{\text{ld}} v_3 \xrightarrow{\times} v_4 \xrightarrow{+} v_5 \xrightarrow{\text{st}}
\end{align*}
\]
Difference Constraints

- A **difference constraint** is a formula in the form of $x - y \leq b$ or $x - y < b$ for numeric variables $x$ and $y$, and constant $b$

- With scheduling variables, we use **integer difference constraints** to model a variety of scheduling constraints
  - $x$ and $y$ must have integral values
    - Thus $b$ only needs to be an integer => form $x - y < b$ is redundant
SDC Constraint Matrix

- The constraint matrix of SDC(X, C) is a totally unimodular matrix (TUM):
  - Every nonsingular square submatrix has a determinant of -1/+1.

\[
\begin{pmatrix}
  1 & 0 & 0 & 0 & -1 & 0 \\
  0 & 1 & 0 & -1 & 0 & 0 \\
  0 & 0 & 1 & -1 & 0 & 0 \\
  0 & 0 & 0 & 1 & -1 & 0 \\
  0 & 0 & 0 & 0 & 1 & -1 \\
  0 & 0 & 1 & 0 & 0 & -1 \\
  0 & 1 & 0 & 0 & 0 & -1 \\
\end{pmatrix}
\begin{pmatrix}
  s_0 \\
  s_1 \\
  s_2 \\
  s_3 \\
  s_4 \\
  s_5 \\
\end{pmatrix}
\leq
\begin{pmatrix}
  0 \\
  0 \\
  0 \\
  0 \\
  0 \\
  -1 \\
\end{pmatrix}
\]

- Theorem (Hoffman & Kruskal, 1956): If A is totally unimodular and b is a vector of integers, every extreme point of polyhedron \{x : Ax \leq b\} is integral.
  - Solving linear programming (LP) relaxation leads to integral solutions
Difference constraints can be conveniently represented using **constraint graph**

- Each vertex represents a variable, and each weighted edge corresponds to a different constraint
- Detect infeasibility by the presence of negative cycle (by solving single-source shortest path)

```
s_0 - s_4 \leq -1
s_4 - s_2 \leq 0
0 \leq -1
```

**SDC Constraint Graph**
Handling Resource Constraints (NP-Hard in General)

- Resource constraints cannot be represented exactly in integer difference form

- Resource constraints
  - Heuristic partial orderings

\[
\begin{align*}
\text{v}_0 \rightarrow \text{v}_2 : s_0 - s_2 & \leq -1 \quad 3 \text{ cycle latency} \\
\text{v}_1 \rightarrow \text{v}_0 : s_1 - s_0 & \leq -1 \\
\text{v}_2 \rightarrow \text{v}_0 : s_2 - s_0 & \leq -1 \quad 2 \text{ cycle latency}
\end{align*}
\]

- Resource constraint
  - Two read ports

[J. Cong & Z. Zhang, DAC, 2006] [Z. Zhang & B. Liu, ICCAD, 2013]
Linear Objectives

- ASAP: \( \min \sum_{i \in V} s_i \)
- ALAP: \( \max \sum_{i \in V} s_i \)
- Minimum latency: \( \min \max_{i \in V} \{s_i\} \)
- Minimum average case latency (control-intensive design)
- Many other ...

- Target cycle time: 5ns
- Delay estimates
  - Mul (x): 3ns
  - Add (+): 1ns
  - Load/Store (ld/st): 1ns
Control Flow Graphs

- Control dependencies can also be honored
  - If \( bb_2 \) is control dependent on \( bb_1 \), the operation nodes of \( bb_2 \) are not allowed to be scheduled before those of \( bb_1 \)
  - Polarize each basic block \( bb_i \) with two scheduling variables (head and tail)
    - \( \forall v \in bb_i \), \( s_h(bb_i) - s_h(v) \leq 0 \)
    - \( \forall v \in bb_i \), \( s_t(v) - s_t(bb_i) \leq 0 \)
  - If \( e_c(bb_i, bb_j) \in E_c \) and \( e_c \) is not a back edge
    - \( s_t(bb_i) - s_h(bb_j) \leq 0 \)
Example: Greatest Common Divisor (GCD)

```
x = in1;
y = in2;
while (x != y) {
    if (x > y)
        x = x - y;
    else
        y = y - x;
}
*out = x;
```
GCD in SSA form

\[
\begin{align*}
x &= \text{in1;} \\
y &= \text{in2;} \\
\text{while (x \neq y) } & \{ \\
\quad \text{if (x > y) } \\
\qquad x &= x - y; \\
\quad \text{else } y &= y - x; \\
\} \\
*\text{out} &= x;
\end{align*}
\]
Interpreting the LP Solution of SDC Scheduling

- Scheduling is performed across basic block boundaries

```
x_0 = in1
y_0 = in2
cond1 = (x_0 != y_0)

x_1 = \Phi(x_0, x_1, x_2)
y_1 = \Phi(y_0, y_1, y_2)
cond2 = (x_1 > y_1)

x_2 = x_1 - y_1
cond3 = (x_2 != y_1)

y_2 = y_1 - x_1
cond4 = (x_1 != y_2)

x_3 = \Phi(x_0, x_1, x_2)
*out = x_3
```
Operations and Predicates

0

\[ x_0 = \text{in1} \]
\[ y_0 = \text{in2} \]
\[ \text{cond1} = (x_0 \neq y_0) \]

Add predicates for conditionally executed operations in each state

1

\[ x_1 = \Phi (x_0, x_1, x_2) \]
\[ y_1 = \Phi (y_0, y_1, y_2) \]
\[ \text{cond2} = (x_1 > y_1) \]
\[ x_2 = x_1 - y_1 \]
\[ \text{cond3} = (x_2 \neq y_1) \]
\[ y_2 = y_1 - x_1 \]
\[ \text{cond4} = (x_1 \neq y_2) \]
\[ x_3 = \Phi (x_0, x_1, x_2) \]
\[ *\text{out} = x_3 \]

if (cond1) {
    \[ x_1 = \Phi (x_0, x_1, x_2) \]
    \[ y_1 = \Phi (y_0, y_1, y_2) \]
    \[ \text{cond2} = (x_1 > y_1) \]
    if (cond2) {
        \[ x_2 = x_1 - y_1 \]
        \[ \text{cond3} = (x_2 \neq y_1) \]
    } else {
        \[ y_2 = y_1 - x_1 \]
        \[ \text{cond4} = (x_1 \neq y_2) \]
    }
}

if (!cond1 || (!cond3 && !cond4)) {
    \[ x_3 = \Phi (x_0, x_1, x_2) \]
    \[ *\text{out} = x_3 \]
}
Deriving State Transition Graph (STG)

Predicates for operations and state transitions can be derived from original control flow and dominance analysis.

$x_0 = \text{in}1$
$y_0 = \text{in}2$
$\text{cond}_1 = (x_0 \neq y_0)$

if (\text{cond}1) {
    x_1 = \Phi (x_0, x_1, x_2)
    y_1 = \Phi (y_0, y_1, y_2)
    \text{cond}2 = (x_1 > y_1)
    \text{if (cond}2) {
        x_2 = x_1 - y_1
        \text{cond}3 = (x_2 \neq y_1)
    } \text{else} {
        y_2 = y_1 - x_1
        \text{cond}4 = (x_1 \neq y_2)
    }
}\text{if (!cond}1 \text{|| (!cond}3 \text{&& !cond}4)) \{
    x_3 = \Phi (x_0, x_1, x_2)
    *\text{out} = x_3
\}

!\text{cond}1 \text{&& (cond}3 \text{|| cond}4)
**Exact Encoding of Resource Constraints**

Difficult to exactly encode resource constraints in the strict SDC form

Two read ports only!

Load operations must be serialized

(NP-Hard in general)

$$s_0 - s_1 \neq 0$$

Using Boolean formulas instead

Resource sharing variable

$v_0$ and $v_1$ share the same port?

$$R_{0,1} \downarrow$$

Ordering variable

$v_0$ scheduled before $v_1$?

$$O_{0 \rightarrow 1}$$

$v_1$ scheduled before $v_0$?

$$O_{1 \rightarrow 0}$$

**Note:** $R_{0,1} \Rightarrow (O_{0 \rightarrow 1} \vee O_{1 \rightarrow 0})$ reads “$R_{0,1}$ implies $O_{0 \rightarrow 1}$ or $O_{1 \rightarrow 0}$”
*SDS: Exact and Practically Scalable Scheduling with SDC and SAT

**Graph based feasibility checking**

**Conflict-driven learning**

Conflict based search

~1M variables

>1M clauses

SDC

Timing Constraints

\[ R_{01} \rightarrow ( O_{0 \rightarrow 1} \lor O_{1 \rightarrow 0} ) \]

\[ \neg ( O_{0 \rightarrow 1} \land O_{1 \rightarrow 0} ) \]

\[ R_{02} \rightarrow ( O_{0 \rightarrow 2} \lor O_{2 \rightarrow 0} ) \]

\[ \neg ( O_{0 \rightarrow 2} \land O_{2 \rightarrow 0} ) \]

\[ R_{12} \rightarrow ( O_{1 \rightarrow 2} \lor O_{2 \rightarrow 1} ) \]

\[ \neg ( O_{1 \rightarrow 2} \land O_{2 \rightarrow 1} ) \]

\[ s_0 - s_4 \leq 0 \]

\[ s_1 - s_3 \leq 0 \]

\[ s_2 - s_3 \leq 0 \]

\[ s_3 - s_4 \leq 0 \]

\[ s_4 - s_5 \leq 0 \]

\[ s_2 - s_5 \leq -1 \]

\[ s_1 - s_5 \leq -1 \]

**SAT**

Resource Constraints

**Conflict based search**

~1M variables

>1M clauses

**Conflict clauses**

**Partial orderings**

**Difference constraints**

Infeasibility

[S. Dai, G. Liu, and Z. Zhang, FPGA 2018]
Scheduling Summary

- **ILP**
  - Exact, but exponential worst-case runtime

- **Hu’s algorithm**
  - Optimal and polynomial
  - Only works in very restricted cases

- **List scheduling**
  - Extension to Hu’s for general cases
  - Greedy (fast) but suboptimal

- **SDC-based scheduling**
  - A versatile heuristic based on LP formulation with different constraints
  - Amenable to global optimization
Next Lecture

- Resource sharing
- Pipelining concepts
Acknowledgements

- These slides contain/adapt materials developed by
  - Ryan Kastner (UCSD)