More Scheduling
Outline

► More on resource-constrained scheduling
  – ILP formulation
  – List scheduling

► Time-constrained scheduling

► Scalable scheduling with realistic design constraints
  – SDC-based scheduling
Exercise: ILP for ASAP scheduling

- Two operations: $v_1$ and $v_2$
  - Each operate has a full-cycle delay
  - No operation chaining allowed
- $L = 3$, i.e., a three-cycle scheduling window
- Objective: ASAP (no resource constraints here)
- Please write down the ILP formulation

\[
\begin{align*}
&v_1 \\
&\times \\
&v_2 \\
&+ \\
\end{align*}
\]
ILP Formulation of RCS

- Linear constraints:
  - Unique start times: \( \sum_{k} x_{ik} = 1, \quad i = 1, 2, ..., N \)
  
  - Dependence must be satisfied (no chaining)
    \[
    t_j \geq t_i + d_i + 1 : \forall (v_i, v_j) \in E \Rightarrow \sum_{k} k x_{jk} \geq \sum_{k} x_{ik} + d_i + 1
    \]
  
  - Resource constraints
    \[
    \sum_{i:RT(v_i)=r} \sum_{l=k-d_i}^{k} x_{il} \leq a_r, \quad r = 1, ..., n_{res}, \quad k = 1, ..., L
    \]

  \( RT(v_i) \) : resource type ID (between 1~n_{res}) of operation \( v_i \)
  \( a_r \) is the number of available resources for resource of type \( r \)
ILP Formulation of RCS: Objective

- Objective: \( \min c^T t \)
  - \( t = \) start times vector, \( c = \) cost weight (e.g., \([0 \ldots 0 1]\))
  - To minimize the overall latency, we can introduce a pseudo node (sink: \( v_{N+1} \)) to serve as a unique output, and use the following objective

\[
\begin{align*}
\min t_{N+1} & \quad \iff \quad \min \sum_{k=1}^{L} k \cdot x_{N+1,k}
\end{align*}
\]
Use of ASAP and ALAP

- In general, the following will help the ILP solver run faster
  - Minimize # of variables and constraints
  - Simplify the constraints

- We can write the ILP without ASAP and ALAP, but using ASAP and ALAP will simplify the inequalities
ILP Formulation: Unique Start Time Constraints

\[ x_{il} = \begin{cases} 0 & \text{for } l < t_i^S \text{ and } l > t_i^L \\ t_i^S = \text{ASAP}(v_i), t_i^L = \text{ALAP}(v_i) \end{cases} \]

- Without using ASAP and ALAP
  \[
x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1
  \]
  \[
x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1
  \]
  \[...\]
  \[
x_{11,1} + x_{11,2} + x_{11,3} + x_{11,4} = 1
  \]

- Using ASAP and ALAP
  \[
x_{1,1} = 1
  \]
  \[
x_{2,1} = 1
  \]
  \[...\]
  \[
x_{6,1} + x_{6,2} = 1
  \]
  \[...\]
  \[
x_{9,2} + x_{9,3} + x_{9,4} = 1
  \]
  \[...\]

assume L=4
ILP Formulation: Dependence Constraints

- Using ASAP and ALAP, the non-trivial inequalities are:
  (assuming no chaining and single-cycle ops)

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

\[\text{assume } L=4\]
ILP Formulation: Resource Constraints

Resource constraints (assuming 2 ALUs and 2 multipliers)

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

assume \( L=4 \)
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 refer 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 MUL Ts, 1 AddSub, and 1 CMP available
A Special Case

- With the following (very) restrictive conditions:
  - All operations have unit delay
  - All operations (and resources) 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
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)
Exercise: ILP for Classroom Allocation

- Minimize the number of classrooms that the school has to 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 # of rooms required

<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
(4) 2:00 – 4:00pm
Summary: ILP Scheduling

- **Pros:** versatile modeling ability
  - Can be extended to handle almost every design aspects
    - Resource allocation
    - Module selection
    - Area, power, etc.

- **Cons:** computationally expensive
  - \#variables = O( \#nodes * \#c-steps)
  - 0-1 assignment variables: need extensive search to find optimal solution
Tension between Scalability and Quality

High scalability (w/ greedy decisions)

List scheduling (e.g., [Parker et al., DAC’86])

High quality (w/ global optimization)

Slow runtime

Low quality

Meta heuristics (e.g., Ant colony [Wang et al., TCAD’07])

Force-directed

[M] ILP

① Handle rich constraints
② Perform global optimization
③ Archive fast runtime

19
SDC-Based Scheduling

- **SDC = System of difference constraints**

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

  - Schedule variables for operation $i$: $s_i$

  - 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
    - $v_2 \rightarrow v_5: s_2 - s_5 \leq -1$
    - $v_1 \rightarrow v_5: s_1 - s_5 \leq -1$

  [J. Cong & Z. Zhang, DAC, 2006] [Z. Zhang & B. Liu, ICCAD, 2013]
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 $\Rightarrow$ form $x - y < b$ is redundant.
Difference constraints can be conveniently represented using constraint graph
- Detect infeasibility by the presence of negative cycle
The constraint matrix of SDC is a **totally unimodular matrix (TUM)**:
- Every nonsingular square submatrix has a determinant of -1/+1.

\[
A = \begin{pmatrix}
1 & 0 & 0 & 0 & -1 & 0 \\
0 & 1 & 0 & -1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 \\
1 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 1 & 0 & 0 & -1 \\
0 & 1 & 0 & 0 & 0 & -1
\end{pmatrix} \quad s = \begin{pmatrix}
s_0 \\
s_1 \\
s_2 \\
s_3 \\
s_4 \\
s_5
\end{pmatrix} \quad \leq \begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
-1
\end{pmatrix} \quad b
\]

Linear programming with a TUM constraint matrix guarantees integral solutions [Hoffman & Kruskal, 1956] [Hochbaum & Shanthikumar, 1990]
Next Class

- More on SDC scheduling
- Resource sharing
Acknowledgements

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