Module 1

◆ Objectives:

• The scheduling problem
  ♦ Case analysis

• Scheduling without constraints
• Scheduling with timing constraints
Scheduling

◆ Circuit model:
  • Sequencing graph
  • Cycle-time is given
  • Operation delays expressed in cycles

◆ Scheduling:
  • Determine the start times for the operations
  • Satisfying all the sequencing (timing and resource) constraint

◆ Goal:
  • Determine area/latency trade-off

Example
Taxonomy

◆ Unconstrained scheduling
◆ Scheduling with timing constraints:
  • Latency
  • Detailed timing constraints
◆ Scheduling with resource constraints
◆ Related problems:
  • Chaining
  • Synchronization
  • Pipeline scheduling

Operation Scheduling

◆ Input:
  • Sequencing graph $G(V, E)$, with $n$ vertices
  • Cycle time $\tau$.
  • Operation delays $D = \{d_i; i=0..n\}$.
◆ Output:
  • Schedule $\phi$ determines start time $t_i$ of operation $v_i$.
  • Latency $\lambda = t_n - t_0$.
◆ Goal: determine area / latency tradeoff
◆ Classes:
  • Non-hierarchical and unconstrained
  • Latency constrained
  • Resource constrained
  • Hierarchical
Simplest method

◆ All operations have bounded delays
◆ All delays are in cycles:
  • Cycle-time is given
◆ No constraints – no bounds on area
◆ Goal:
  • Minimize latency

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 \( \phi: V \rightarrow \mathbb{Z}^+ \) 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
◆ 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 $t_n=\lambda$.
- 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 $v_i$
Remarks

◆ ALAP solves a latency-constrained problem
◆ Latency bound can be set to latency computed by ASAP algorithm
◆ Mobility:
  • Defined for each operation
  • Difference between ALAP and ASAP schedule
◆ Slack on the start time

Example

◆ Operations with zero mobility:
  • \{v_5, v_3, v_4, v_5\}
  • Critical path
◆ Operations with mobility one:
  • \{v_9, v_7\}
◆ Operations with mobility two:
  • \{v_9, v_9, v_{33}, v_{71}\}
Scheduling under detailed timing constraints

◆ Motivation:
  • Interface design
  • Control over operation start time

◆ Constraints:
  • Upper/lower bounds on start-time difference of any operation pair

◆ Feasibility of a solution

Constraint graph model

◆ Start from sequencing graph
  • Model delays as weights on edges

◆ Add forward edges for minimum constraints:
  • Edge \((v_i, v_j)\) with weight \(l_{ij} \rightarrow t_j \geq t_i + l_{ij}\)

◆ Add backward edges for maximum constraints:
  • That is, for constraint from \(v_i\) to \(v_j\)
    add backward edge \((v_j, v_i)\) with weight: \(-u_{ij}\)
    • because \(t_j \leq t_i + u_{ij} \rightarrow t_j \geq t_i - u_{ij}\)
Example

Methods for scheduling under detailed timing constraints

- **Assumption:**
  - All delays are fixed and known
- **Set of linear inequalities**
- **Longest path problem**
- **Algorithms:**
  - Bellman-Ford, Liao-Wong
- **Extensions:**
  - Unbounded delays, relative scheduling
Method for scheduling with unbounded-delay operations

- Unbounded delays:
  - Synchronization
  - Unbounded-delay operations (e.g. loops)
- Anchors:
  - Unbounded-delay operations
- Relative scheduling:
  - Schedule ops w.r. to the anchors
  - Combine schedules

Example

- \( t_3 = \max \{ t_1 + d_1; t_a + d_a \} \)
Relative scheduling method

◆ For each vertex:
  - Determine relevant anchor set $R(v_i)$
  - Anchors affecting start time
  - Determine time offsets from anchors

◆ Start-time:
  - Expressed by: $t_i = \max \{ t_a + d_a + t_l \}$
  - Computed only at run-time because delays of anchors are unknown

Relative scheduling under timing constraints

◆ Problem definition:
  - Detailed timing constraints
  - Unbounded delay operations

◆ Solution:
  - May or may not exist
  - Problem may be ill-specified
Relative scheduling under timing constraints

◆ Feasible problem:
  • A solution exists when unknown delays are zero

◆ Well-posed problem:
  • A solution exists for any value of the unknown delays

◆ Theorem:
  • A constraint graph can be well-posed if there are no cycles with unbounded weights

Example

(a)  
(b)  
(c)
Relative scheduling approach

◆ Analyze graph:
  • Detect anchors
  • Well-posedness test
  • Determine dependencies from anchors

◆ Schedule ops with respect to relevant anchors:
  • Bellman-Ford, Liao-Wong, Ku algorithms

◆ Combine schedules to determine start times:
  \[ t_i = \max \{ t_a + d_a + t_j \} \]
  \( a \in R(v_i) \)

Example

| Vertex | Relevant Anchor Set | Offsets | | --- | --- | --- |
| --- | --- | --- |
| \( v_i \) | \( R(v_i) \) | \( t_a \) | \( t_j \) |
| \( a \) | \( \{v_0\} \) | 0 | - |
| \( v_1 \) | \( \{v_0\} \) | 0 | - |
| \( v_2 \) | \( \{v_0\} \) | 2 | - |
| \( v_3 \) | \( \{v_0, a\} \) | 3 | 0 |
Module 2

◆ Objectives:
  - Scheduling with resource constraints
  - Exact formulation:
    ◆ ILP
    ◆ Hu's algorithm
  - Heuristic methods
    ◆ List scheduling
    ◆ Force-directed scheduling
Scheduling under resource constraints

- Classical scheduling problem:
  - Fix area bound – minimize latency (ML-RCS)
- The amount of available resources affects the achievable latency
- Dual problem:
  - Fix latency bound – minimize resources (MR-LCS)
- Assumption:
  - All delays bounded and known

Minimum latency resource-constrained scheduling (ML-RCS)

- Given a set of ops $V$ with integer delays $D$, a partial order on the operations $E$, and upper bounds $\{a_{k,i} : k = 1, 2, \ldots, n_{res}\}$ on resource usage:
- Find an integer labeling of the operation $\varphi : V \rightarrow \mathbb{Z}^+$ such that:
  - $t_i = \varphi(v_i)$,
  - $t_i \geq t_j + d_j$ for all $i,j$ s.t. $(v_j, v_i) \in E$,
  - $| \{ v_i \mid T(v_i) = k \text{ and } t_i \leq l < t_j + d_j \} | \leq a_k$ for all types $k = 1, 2, \ldots, n_{res}$ and steps $l$

  and $t_n$ is minimum
Scheduling under resource constraints

◆ Intractable problem
◆ Algorithms:
  • Exact:
    ◆ Integer linear program
    ◆ Hu (restrictive assumptions)
  • Approximate:
    ◆ List scheduling
    ◆ Force-directed scheduling

ILP formulation

◆ Binary decision variables:
  \( X = \{ x_{il} \mid i = 1,2,\ldots, n; \ l = 1,2,\ldots, \lambda + 1 \} \)
  \( x_{il} \) is TRUE only when operation \( v_i \) starts in step \( l \) of the schedule
  (i.e. \( l = t_i \))
  \( \lambda \) is an upper bound on latency

◆ Start time of operation \( v_i \):
  \( \Sigma_l \ l \cdot x_{il} \)
ILP formulation constraints

◆ Operations start only once
\[ \sum x_{il} = 1 \quad i = 1, 2, \ldots, n \]

◆ Sequencing relations must be satisfied
\[ t_i \geq t_j + d_j \quad \rightarrow \quad t_i - t_j - d_j \geq 0 \quad \text{for all } (v_j, v_i) \in E \]
\[ \sum l \cdot x_{il} - \sum l \cdot x_{jl} - d_j \geq 0 \quad \text{for all } (v_j, v_i) \in E \]

◆ Resource bounds must be satisfied
Simple case (unit delay)
\[ \sum l \cdot x_{il} \leq a_k \quad k = 1, 2, \ldots, n_{res} \quad \text{for all } l \quad i: T(v_i) = k \]

Start Time vs. Execution Time

◆ For each operation \( v_i \), only 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?
      \[ \sum_{m=l-d_i+1}^{l} x_{im} \geq 1 \]
Operation $v_i$ Still Running at Step $l$?

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

  \[
  \begin{array}{ccc}
  & 4 & 4 \\
  & 5 & 5 \\
  & 6 & 6 \\
  & v_9 & v_9 \\
  x_{9,6} = 1 & x_{9,5} = 1 & x_{9,4} = 1
  \end{array}
  \]

- 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

---

Operation $v_i$ Still Running at Step $l$?

- Is $v_i$ running at step $l$?
  - Is $x_{i,l} + x_{i,l-1} + \ldots + x_{i,l-d_i+1} = 1$?

  \[
  \begin{array}{ccc}
  & l-d_i+1 & l-d_i+1 \\
  & \vdots & \vdots \\
  & l-1 & l-1 \\
  & l & l \\
  & v_i & v_i \\
  x_{i,l} = 1 & x_{i,l-1} = 1 & x_{i,l-d_i+1} = 1
  \end{array}
  \]
ILP Formulation of ML-RCS

- **Constraints:**
  - Unique start times: \( \sum_l x_{il} = 1, \quad i = 0, 1, \ldots, n \)
  - Sequencing (dependency) relations must be satisfied
    \( t_i \geq t_j + d_j \quad \forall (v_j, v_i) \in E \Rightarrow \sum_l l \cdot x_{il} \geq \sum_l l \cdot x_{jl} + d_j \)
  - Resource constraints
    \( \sum_{i:T(v_i)=k} \sum_{m=l-d_j+1}^l x_{im} \leq a_k, \quad k = 1, \ldots, n_{res}, \quad l = 1, \ldots, \bar{\lambda} + 1 \)

- **Objective:** \( \min c^T t \).
  - \( t = \) start times vector, \( c = \) cost weight (e.g., \( [0 \ 0 \ \ldots \ 1] \))
  - When \( c = [0 \ 0 \ \ldots \ 1], \ c^T t = \sum_l l \cdot x_{nl} \)

Example

- **Resource constraints:**
  - 2 ALUs; 2 Multipliers
  - \( a_1 = 2; \ a_2 = 2 \)
- **Single-cycle operation**
  - \( d_i = 1 \) for all \( i \)
ILP Example

- Assume $\lambda = 4$
- First, perform ASAP and ALAP
  - (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:

  $x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1$
  $x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1$
  ...
  ...
  ...
  $x_{11,1} + x_{11,2} + x_{11,3} + x_{11,4} = 1$

- Using ASAP and ALAP:

  $x_{1,1} = 1$
  $x_{2,1} = 1$
  $x_{3,2} = 1$
  $x_{4,3} = 1$
  $x_{5,4} = 1$
  $x_{6,1} + x_{6,2} = 1$
  $x_{7,2} + x_{7,3} = 1$
  $x_{8,1} + x_{8,2} + x_{8,3} = 1$
  $x_{9,2} + x_{9,3} + x_{9,4} = 1$
  ....
ILP Example: Dependency Constraints

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

  \[
  2 \cdot x_{7,2} + 3 \cdot x_{7,3} - x_{6,1} - 2 \cdot x_{6,2} - 1 \geq 0
  \]

  \[
  2 \cdot x_{9,2} + 3 \cdot x_{9,3} + 4 \cdot x_{9,4} - x_{8,1} - 2 \cdot x_{8,2} - 3 \cdot x_{8,3} - 1 \geq 0
  \]

  \[
  2 \cdot x_{11,2} + 3 \cdot x_{11,3} + 4 \cdot x_{11,4} - x_{10,1} - 2 \cdot x_{10,2} - 3 \cdot x_{10,3} - 1 \geq 0
  \]

  \[
  4 \cdot x_{5,4} - 2 \cdot x_{7,2} - 3 \cdot x_{7,3} - 1 \geq 0
  \]

  \[
  5 \cdot x_{n,5} - 2 \cdot x_{9,2} - 3 \cdot x_{9,3} - 4 \cdot x_{9,4} - 1 \geq 0
  \]

  \[
  5 \cdot x_{n,5} - 2 \cdot x_{11,2} - 3 \cdot x_{11,3} - 4 \cdot x_{11,4} - 1 \geq 0
  \]

ILP Example: Resource Constraints

- Resource constraints (assuming 2 adders and 2 multipliers)

  \[
  x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} \leq 2
  \]

  \[
  x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} \leq 2
  \]

  \[
  x_{7,3} + x_{8,3} \leq 2
  \]

  \[
  x_{10,1} \leq 2
  \]

  \[
  x_{9,2} + x_{10,2} + x_{11,2} \leq 2
  \]

  \[
  x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} \leq 2
  \]

  \[
  x_{5,4} + x_{9,4} + x_{11,4} \leq 2
  \]

- Objective:

  - Since \( \lambda = 4 \) and sink has no mobility, any feasible solution is optimum, but we can use the following anyway:

  \[
  \text{Min} \quad x_{n,1} + 2 \cdot x_{n,2} + 3 \cdot x_{n,3} + 4 \cdot x_{n,4}
  \]
Minimize resource usage under latency constraint

Additional constraint:
- Latency bound must be satisfied
- $\sum l_i x_{nl} \leq \lambda + 1$

Resource usage is unknown in the constraints
Resource usage is the objective to minimize
**Example**

- Multiplier area = 5
- ALU area = 1.
- Objective function: $5a_1 + a_2$

**ILP Solution**

- Use standard ILP packages
- Transform into LP problem
- Advantages:
  - Exact method
  - Others constraints can be incorporated
- Disadvantages:
  - Works well up to few thousand variables
Hu’s Algorithm

◆ Simple case of the scheduling problem
  • Operations of unit delay
  • Operations (and resources) of the same type

◆ Hu’s algorithm
  • Greedy, polynomial AND optimal (exact)
  • Computes lower bound on number of resources for given latency
    OR
    Computes lower bound on latency subject to resource constraints

◆ Basic idea:
  • Label operations based on their distances from the sink
  • Try to schedule nodes with higher labels first
    (i.e., most “critical” operations have priority)

Hu’s algorithm with $\bar{a}$ resources

◆ Label operations with distance to sink
◆ Set step $l = 1$
◆ Repeat until all ops are scheduled:
  • $U =$ unscheduled vertices in $V$
    • predecessors have been scheduled (or no predecessors)
  • Select $S \subseteq U$ resources with
    • $|S| \leq \bar{a}$
    • Maximal labels
  • Schedule the $S$ operations at step $l$
  • Increment step $l = l + 1$
Example

Assumptions:
- One resource type only
- All operations have unit delay

Labels:
- Distance to sink

Step 1: Op 1,2,6
Step 2: Op 3,7,8
Step 3: Op 4,9,10
Step 4: Op 5,11
List scheduling algorithms

- **Heuristic method for:**
  - Min latency subject to resource bound (ML-RCS)
  - Min resource subject to latency bound (MR-LCS)

- **Greedy strategy (like Hu’s)**
  - Does not guarantee optimality (unlike Hu’s)

- **General graphs (unlike Hu’s)**
  - Resource constraints on different resource types
  - Operations of arbitrary delay

- **Priority list heuristics**
  - Priority decided by criticality (similar to Hu’s)
  - Longest path to sink, longest path to timing constraint
  - \(O(n)\) time complexity

---

List scheduling algorithm for minimum latency

```
LIST_L( G(V, E), a ) {
    i = 1;
    repeat {
        for each resource type k = 1, 2, ..., nres {
            Determine ready operations \(U_{i,k}\);
            Determine unfinished operations \(T_{i,k}\);
            Select \(S_k \subseteq U_{i,k}\) vertices, s.t. \(|S_k| + |T_{i,k}| \leq a_k\);
            Schedule the \(S_k\) operations at step \(i\);
            \(i = i + 1;\)
        }
    } until \(\{v_o\} \text{ is scheduled} \);
    return \(t;\);
}
```
Resource bounds:
- 3 multipliers with delay 2
- 1 ALU with delay 1

List scheduling algorithm for minimum resource usage

```
LIST_R( G(V, E), λ) {
    a = 1;
    Compute the latest possible start times \( t^L \) by ALAP ( G(V, E), λ);
    if (\( t^L_0 < 0 \))
        return (Ø);
    \( l = 1 \);
    repeat \{ 
        for each resource type \( k = 1, 2, \ldots, n_{res} \) \{
            Determine ready operations \( U_{l,k} \);
            Compute the slacks \( s_i = t_i - l \) for all \( v_i \in U_{l,k} \);
            Schedule the candidate operations with zero slack and update \( a \);
            Schedule the candidate operations not needing additional resources;
        \}
        \( l = l + 1 \);
    \} until (\( v_n \) is scheduled);
    return (t, a);
}
```

(c) Giovanni De Micheli
**Example**

Assumptions
- Unit-delay resources
- Maximum latency = 4

Start with:
- \( a_1 = 1 \) multiplier
- \( a_2 = 1 \) ALUs

---

**Force-Directed Scheduling**

- Heuristic, similar to list scheduling
  - Can handle ML-RCS and MR-LCS
  - For ML-RCS, schedules step-by-step
  - BUT, selection of the operations tries to find the *globally* best set of operations

- Idea [Paulin]
  - Find the mobility \( \mu_i = t_i^L - t_i^S \) of operations (ALAP-ASAP)
  - Look at the operation type probability distributions
  - Try to flatten the operation type distributions

- Definition: operation probability density
  - \( p_i(l) = \Pr \{ v_i \text{ executes in step } l \} \)
  - Assume uniform distribution: \( p_i(l) = \frac{1}{\mu_i + 1} \) for \( l \in [t_i^S, t_i^L] \)
Force-Directed Scheduling: Definitions

- Operation-type distribution (sum of operation probabilities for each type)
  \[ q_k(l) = \sum_{i: T(v_i) = k} p_i(l) \]

- Operation probabilities over control steps:
  \[ p_i = \{p_i(0), p_i(1), \ldots, p_i(n)\} \]

- Distribution graph of type \( k \) over all steps:
  \[ \{q_k(0), q_k(1), \ldots, q_k(n)\} \]

  - \( q_k(l) \) can be thought of as expected operator cost for implementing operations of type \( k \) at step \( l \).

Example

\[
\begin{align*}
q_{\text{add}} (1) &= \frac{1}{3} = 0.33 \\
q_{\text{add}} (2) &= \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1 \\
q_{\text{add}} (3) &= 1 + \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 2 \\
q_{\text{add}} (4) &= 1 + \frac{1}{3} + \frac{1}{3} = 1.66 \\
q_{\text{sub}} (1) &= 1 + 1 + \frac{1}{2} + \frac{1}{3} = 2.83 \\
q_{\text{sub}} (2) &= 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{3} = 2.33 \\
q_{\text{sub}} (3) &= \frac{1}{2} + \frac{1}{3} = 0.83 \\
q_{\text{sub}} (4) &= 0
\end{align*}
\]

© K. Bazargan
Force-Directed Scheduling Algorithm

- Very similar to LIST_L(G(V,E), a)
  - Compute mobility of operations using ASAP and ALAP
  - Computer operation probabilities and type distributions
  - Select and schedule operations
  - Update operation probabilities and type distributions
  - Go to next control step

- Difference with list scheduling in selecting operations
  - Select operations with least force
  - $O(n^2)$ time complexity due to pair-wise force computations

Force

- Used as priority function

- Force is related to concurrency:
  - Sort operations for least force

- Mechanical analogy:
  - Force = constant x displacement
    - Constant = operation-type distribution
    - Displacement = change in probability
Two Types of Forces

◆ Self-force:
  • Sum of forces to feasible schedule steps
  • Self-force for operation $v_i$ in step $l$
    ◦ Sum over type distribution x delta probability
    $$\sum_{m \in \text{interval}} q_i(m) (\delta_l - p_i(m))$$
    ◦ Higher self-force indicates higher mobility

◆ Predecessor/successor-force:
  • Related to the predecessors/successors
    ◦ Fixing an operation timeframe restricts timeframe of predecessors/successors
    ◦ Ex: Delaying an operation implies delaying its successors
    ◦ Computed by changes in self-forces of neighbors

Example: Schedule operation $v_6$

Operation $v_6$ can be scheduled in step 1 or step 2
Example: operation $v_6$

- $v_6$ can be scheduled in the first two steps
  - $p(1) = 0.5; p(2) = 0.5; p(3) = 0; p(4) = 0$
- Distribution: $q(1) = 2.8; q(2) = 2.3$

- Assign $v_6$ to step 1:
  - variation in probability $1 - 0.5 = 0.5$ for step 1
  - variation in probability $0 - 0.5 = -0.5$ for step 2
- Self-force: $2.8 \cdot 0.5 - 2.3 \cdot 0.5 = +0.25$
- No successor force

Example: operation $v_6$

- Assign $v_6$ to step 2:
  - variation in probability $0 - 0.5 = -0.5$ for step 1
  - variation in probability $1 - 0.5 = 0.5$ for step 2
- Self-force: $-2.8 \cdot 0.5 + 2.3 \cdot 0.5 = -0.25$
- Successor-force:
  - Operation $v_7$ assigned to step 3
  - Succ. force is $2.3 (0 - 0.5) + 0.8 (1 - 0.5) = -0.75$
- Total force = -1
Example: operation $v_6$

- Total force in step 1 = + 0.25
- Total force in step 2 = -1

Conclusion:
- Least force is for step 2
- Assigning $v_6$ to step 2 reduces concurrency

Force-directed scheduling algorithm for minimum resources

```
FDS ( G ( V, E ), $\lambda$ ) {
    repeat {
        Compute/update the time-frames;
        Compute the operation and type probabilities;
        Compute the self-forces, p/s-forces and total forces;
        Schedule the op. with least force;
    } until (all operations are scheduled)
    return (t);
}
```
Scheduling Generalizations

- Conditional operations
- Hierarchy
- Resource generalizations
  - Multi-cycling and chaining
  - Pipelined resources
- Model generalizations
  - Pipelining
  - Loops

Multi-Cycling and Chaining

- Consider propagation delays of resources not in terms of cycles
- Use scheduling to chain multiple operations in the same control step
- Useful technique to explore effect of cycle-time on area/latency trade-off
- Algorithms:
  - ILP, ALAP/ASAP, list scheduling
Example

Cycle-time: 60

Pipelining

Two levels of pipelining:

- Structural pipelining
  - Pipelined resources
  - Non-pipelined model
- Functional pipelining
  - Non-pipelined resources
  - Pipelined model
Structural Pipelining

- Non-pipelined model using pipelined resources
- Resources characterized by
  - Execution delay
  - Data introduction interval: $DII$
- Implications
  - Operations sharing a pipelined resource are serialized (always)
  - Operations do not have data dependency
- Solution using list scheduling
  - Relax criteria for selection of vertices

Structural Pipelining Example

- 3 multipliers w/ 2 cycle delay and $DII = 1$
Functional Pipelining

- Pipelined model, non-pipelined resources
- Assume non-hierarchical graphs
- Model characterized by
  - Latency
  - Initiation interval, II
- Restart source before completing sink
  - Implicit loop
- Solutions using ILP or heuristics
  - ILP resource constraints modified to include increased concurrency
  - List or force-directed methods

Pipelining and concurrency

- II determines resource usage
  - Smaller II leads to larger overlaps, higher resource requirements
    \[ \min(a_s) = n_k \text{ for } II = 1 \] (all \( n_k \) operations are concurrent)
  - In general, \[ \alpha_k = \left\lfloor \frac{n_k}{II} \right\rfloor \]
- Concurrent operations
  - Operations \( v_i \) and \( v_j \) are executing concurrently at control step \( l \), if
    \[ \text{rem}(t_i/II) = \text{rem}(t_j/II) = l \]
  - Affects the design of the controller circuitry
Loop Scheduling

◆ Potential parallelism across loop invocations

◆ Single loop executions
  • Sequential execution
  • Loop unrolling (known iteration count)
    ♦ Merge multiple iterations into one to provide scheduling opportunities
  • Loop pipelining (iteration count might be unknown)
    ♦ Start next iteration while current one is still running
    ♦ Depends on dependencies across iterations

◆ Merging of multiple loops
  • Run different loops in parallel (no dependencies)

Loop Scheduling Example

◆ Sequential

```
1 2 3 4 5 6 7 8
```

◆ Unrolled

```
1,2,3 4,5,6 7,8,9
```

◆ Pipelined

```
1 3 5 7
2 4 6 8
```
### Loop Pipelining

- Iteration count = \( N \)
- Loop latency = \( N \cdot \lambda \)
- Pipeline loop iterations with \( II < \lambda \)
- Latency of the pipelined loop = \( N \cdot II + \text{overhead} \)
- Overhead = \( \left\lfloor \frac{\lambda}{II} \right\rfloor - 1 \)

---

### Summary

- Scheduling determines area/latency trade-off
- Intractable problem in general:
  - Heuristic algorithms
  - ILP formulation (small-case problems)
- Several heuristic formulations
  - List scheduling is the fastest and most used
  - Force-directed scheduling tends to yield good results
- Several extensions
  - Chaining and multi-cycling
  - Pipelining