Scheduling

- What have we done?
  - Specified the order in which operations are performed
- What’s next - Operator Scheduling
  - Assigning operations performed in each state
  - Generally speaking, determining the start time of each task/operation
- Why is scheduling important?
  - Determines the amount of concurrency of the resulting implementation – affects performance
  - Maximum amount of concurrent operations of a given type at any time step also determines the amount of hardware resources of that type required – affects area

Control/Data flow graph (CDFG)

- How do we represent a schedule?
- Dataflow graph – represents the way data flows through a computation
  - Computations limited to 2-input

Code fragment we want to implement

\[
\begin{align*}
x_1 &= x + dx; \\
u_1 &= u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx) \\
y_1 &= y + u \cdot dx \\
c &= x_1 < a
\end{align*}
\]

Corresponding CDFG

- Vertices correspond to the operations we perform
- Edges correspond to the dependencies

Task Representation - Sequencing graph

- Determining a schedule
  - Don’t really care about the actual input/output values
  - Just want to know the task we perform (add, subtract, compare, etc.) and the dependencies among the task
- Utilize a Sequencing graph
Sequencing graph

- Source node: Represented by a NOP (no operation) node. Indicates start of computation.
- Sink node: Represented by a NOP (no operation) node. Indicates completion of computation.
- Vertices: Represents the task or operation to be performed.
- Edges: Represents dependencies among operations. Cannot perform operation 3 until operations 1 and 2 are completed.
- Dotted lines: Does NOT represent a dependency between tasks or operations. Represents a connection between the source and the initial task or operations.
- Dotted lines: Does NOT represent a dependency between tasks or operations. Represents a connection between the sink and the final task or operations.

Scheduling - Sequencing Graphs

- Sequencing graph itself only specifies the dependencies among tasks.
- Scheduling requires we associate a start time for each task/operation in the sequencing graph.
- Introduce concept of time:

## ASAP Scheduling

- Unconstrained minimum-latency scheduling problem:
  - We have infinite resources, all we want is the minimum time to perform the computation.
  - Commonly referred to as ASAP (as soon as possible) scheduling.

### ASAP( G(V,E) ):
- Schedule v₀ by setting t₀ = 1
- Repeat:
  - Select a vertex vᵢ whose predecessors are all scheduled;
  - Schedule vᵢ by setting tᵢ = max tⱼ + dⱼ, j ∈ predecessors(vᵢ)
- Until (vₙ is scheduled);
- Return t;

### Example 1
- Step 1: Schedule v₀ at time 1
- Step 2: Select a vertex vᵢ whose predecessors are all scheduled
- Step 3: Schedule vᵢ to time = predecessor's scheduled time + time required for predecessor to execute
- Step 4: Has vᵢ been scheduled yet?
- All of vᵢ predecessors are scheduled
- Notice all operations require only 1 clock cycle to execute.
**ASAP Scheduling**

**Example 1**

**Step 2**
Select a vertex $v_i$ whose predecessors are all scheduled

**Step 3**
Schedule $v_i$ to time = predecessor’s scheduled time + time required for predecessor to execute

**Step 4**
Has $v_i$ been scheduled yet?

---

**Example 1**

**Step 2**
Select a vertex $v_i$ whose predecessors are all scheduled

**Step 3**
Schedule $v_i$ to time = predecessor’s scheduled time + time required for predecessor to execute

**Step 4**
Has $v_i$ been scheduled yet?

---

**Example 1**

**Step 2**
Select a vertex $v_i$ whose predecessors are all scheduled

**Step 3**
Schedule $v_i$ to time = predecessor’s scheduled time + time required for predecessor to execute

**Step 4**
Has $v_i$ been scheduled yet?

---

**Example 1**

**Step 2**
Select a vertex $v_i$ whose predecessors are all scheduled

**Step 3**
Schedule $v_i$ to time = predecessor’s scheduled time + time required for predecessor to execute

**Step 4**
Has $v_i$ been scheduled yet?

---
ASAP Scheduling
Example 1

Step 2
Select a vertex \( v \) whose predecessors are all scheduled

Step 3
Schedule \( v \) to time = predecessor’s scheduled time + time required for predecessor to execute

Step 4
Has \( v \) been scheduled yet? No. Repeat loop.

Step 2
Select a vertex \( v \) whose predecessors are all scheduled

Step 3
Schedule \( v \) to time = predecessor’s scheduled time + time required for predecessor to execute

Step 4
Has \( v \) been scheduled yet? No. Repeat loop.
ASAP Scheduling

Example 1

Step 2: Select a vertex \( v_i \) whose predecessors are all scheduled

Step 3: Schedule \( v_i \) to time = predecessor's scheduled time + time required for predecessor to execute

Step 4: Has \( v_i \) been scheduled yet?

ASAP Scheduling goal is to schedule tasks/operations to perform as soon as possible

We can skip the algorithm and visually move vertices "up" as far as possible

Example 2

Step 2: Select a vertex \( v_i \) whose predecessors are all scheduled

Step 3: Schedule \( v_i \) to time = predecessor's scheduled time + time required for predecessor to execute

Step 4: Has \( v_i \) been scheduled yet?

Yes. We are done!
ASAP Scheduling

Example 3

```
4 5

10

6

1

11

9

7

+/

TIME 1

TIME 2

TIME 3

TIME 4
```

**ALAP Scheduling**

- Latency constrained scheduling problem
  - Schedule must satisfy an upper bound on latency
  - Commonly referred to as ALAP (as late as possible) scheduling

```
ALAP( G(V,E), λ) {
  Schedule v_n by setting t_n = λ + 1
  repeat{
    Select a vertex v_i whose successors are all scheduled;
    Schedule v_i by setting t_i = min { t_j - d_j | (v_j, v_i) ∈ E }
  }
  until (v_0 is scheduled);
  return t;
}
```

ALAP Scheduling

Example 1

ALAP Scheduling goal is to schedule tasks/operations to perform as late as possible

We can skip the algorithm and visually move vertices “down” as far as possible

```
1

2

3

4

5

6

7

8

9

10

11
```

ALAP Scheduling

Example 2

```
1

2

3

4

5

6

7

8

9

10

11
```

ALAP Scheduling

Example 3

```
4 5

10

6

1

11

9

7

+/

TIME 1

TIME 2

TIME 3

TIME 4
```
**Mobility**

- Mobility (or slack) important quantity used by scheduling algorithms
- Mobility = start time \( \text{ALAP scheduling} \) – start time \( \text{ASAP scheduling} \)
- Mobility = 0, task/operation can only be started at the given time in order to meet overall latency constraint
- Mobility > 0, indicates span of possible start times
  - Helps with minimizing resources (adders, multipliers, etc.)

**Example 1 – ASAP Schedule**

```
V_1 mobility = \text{time}_{\text{ASAP}}(V_1) - \text{time}_{\text{ALAP}}(V_1) = 1 - 1 = 0
V_6 mobility = \text{time}_{\text{ASAP}}(V_6) - \text{time}_{\text{ALAP}}(V_6) = 2 - 1 = 1
V_{11} mobility = \text{time}_{\text{ASAP}}(V_{11}) - \text{time}_{\text{ALAP}}(V_{11}) = 4 - 2 = 2
```

**Example 1 – ALAP Schedule**

```
V_1 mobility = \text{time}_{\text{ALAP}}(V_1) - \text{time}_{\text{ASAP}}(V_1) = 1 - 1 = 0
V_6 mobility = \text{time}_{\text{ALAP}}(V_6) - \text{time}_{\text{ASAP}}(V_6) = 2 - 1 = 1
V_{11} mobility = \text{time}_{\text{ALAP}}(V_{11}) - \text{time}_{\text{ASAP}}(V_{11}) = 4 - 2 = 2
```

**Example 1 – ASAP Only**

- What do we get with the ASAP Schedule?
  - Latency = 4
  - Resource requirement = 4 multipliers, 2 ALUs

**Example 1 – ALAP Only**

- What do we get with the ALAP Schedule?
  - Latency = 4
  - Resource requirement = 2 multipliers, 3 ALUs
Mobility

Example 1 – Modify ALAP

- Start with ALAP schedule
- Use mobility to try to improve resource requirements

Operations with mobility = 0
\( v_1, v_2, v_3, v_4, v_5 \)

Operations with mobility = 1
\( v_6, v_7 \)

Operations with mobility = 2
\( v_8, v_9, v_{10}, v_{11} \)

- Start with ALAP schedule
- Use mobility to try to improve resource requirements
- Vertices with mobility = 0 cannot be moved, they are part of the critical path
- Vertices with mobility > 0 can be moved to minimize resource requirements

Resource Constrained Scheduling

- Resource constrained scheduling problem
- Resource usage determines circuit area
- Consider area/latency tradeoff

ASAP schedule determines the minimum latency, we assumed infinite resources

We can determine a schedule to consider only minimizing resources – assuming latency doesn’t matter

Likely we want something in between
Hu’s Algorithm

- Exact (polynomial-time) algorithm for resource constrained scheduling
  - Assumes one resource handles all possible operations
  - Assumes all operations have 1 unit delay

\[
\text{HU}(G(V,E), a) \}
\]

Label the vertices;
\( l = 1; \)
repeat {  
\( U = \) unscheduled vertices in \( V \) without predecessors or whose predecessors have been scheduled;
Select \( S \subseteq U \) vertices, such that \(|S| \leq a \) and labels in \( S \) are maximal;
Schedule the \( S \) operations at step \( l \) by setting \( t = l \cdot i_v \); \( v \in S; \)
\( l = l + 1; \)
} until \( (v_n \text{ is scheduled}); \)
return \( t; \)

Value of \( a \) indicates the number of resources we have available
Label with distance of vertices to sink node
Indicates the time step
Make a list of all vertices not waiting on another operation to be scheduled
Select a subset of the vertices in \( U \), no more than \( a \), choosing vertices with largest labels
Keep going until we have scheduled the sink node \( v_n \)

Hu’s Algorithm
Example 1

\[
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9 \\
10 \\
11 \\
V_0 \\
V_n \\
\end{array}
\]

Step 1
Label all vertices with distance to sink node

Step 2
\( l = 1 \)

Step 3
\( U = \) unscheduled vertices in \( V \) without predecessors or whose predecessors have been scheduled

Step 4
\( S \subseteq U \) vertices, such that \(|S| \leq a \) and labels in \( S \) are maximal

Step 5
Schedule the \( S \) operations at step \( l \) by setting \( t = l \cdot i_v \); \( v \in S; \)

Step 6
\( l = l + 1 \)

Step 7
Has \( v_n \) been scheduled yet?

No. Repeat loop.

Example 1

\[
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9 \\
10 \\
11 \\
V_0 \\
V_n \\
\end{array}
\]

Step 1
Label all vertices with distance to sink node

Step 2
\( l = 1 \)

Step 3
\( U = \) unscheduled vertices in \( V \) without predecessors or whose predecessors have been scheduled

Step 4
\( S \subseteq U \) vertices, such that \(|S| \leq a \) and labels in \( S \) are maximal

Step 5
Schedule the \( S \) operations at step \( l \) by setting \( t = l \cdot i_v \); \( v \in S; \)

Step 6
\( l = l + 1 \)

Step 7
Has \( v_n \) been scheduled yet?

No. Repeat loop.
Hu's Algorithm

Example 1

Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

Example 2

Step 1
Label all vertices with distance to sink

Step 2

Hu’s Algorithm
Example 2

Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

No. Repeat loop.

Example 2

Step 3
U = {v_1, v_2, v_3, v_4, v_7, v_8}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

No. Repeat loop.

Example 2

Step 3
U = {v_5, v_6, v_7, v_8}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

No. Repeat loop.

Example 2

Step 3
U = {v_9, v_10}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

No. Repeat loop.
Hu’s Algorithm

Example 2

Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

Yes. We are done.

Step 1
U = \{ v_n \}

Step 2
S = \{ v_n \}

Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?


Hu’s Algorithm

Example 3

Step 1
Label all vertices with distance to sink

Step 2
l = 1

Step 3
U = \{ v_1, v_2, v_3, v_4, v_7, v_8 \}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

No. Repeat loop.

Step 1

U = \{ v_3, v_4, v_5, v_7, v_8 \}

Step 2

l = 2

Step 3

U = \{ v_3, v_4, v_5, v_7, v_8 \}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

No. Repeat loop.
Hu’s Algorithm

Example 3

Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

---

Step 3
U = \{ v_5, v_6, v_7, v_8 \}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

---

Step 3
U = \{ v_7, v_8, v_9 \}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

---

Step 3
U = \{ v_9, v_{10} \}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?

---

Step 3
U = \{ v_11 \}

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

Step 5
Schedule vertices in S to time step I

Step 6
I = I + 1

Step 7
Has v_n been scheduled yet?
Hu’s Algorithm

Example 3

**Step 3**
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled

**Step 4**
S = subset set of vertices in U, no more than a, where labels are maximal

**Step 5**
Schedule vertices in S to time step I

**Step 6**
I = I + 1

**Step 7**
Has v_n been scheduled yet?

**Additional Scheduling Considerations**

- Hu's algorithm
  - Assumes one resource handles all possible operations
  - Assumes all operations have 1 unit delay

- Most scheduling problems have additional considerations
  - What happens when we have more than one type of task/operation?
  - What happens when a task/operation takes more than 1 unit delay?

- Increased problem space, difficult problem to solve efficiently
  - Many heuristics have been developed to address these problems
    - Minimum-latency, resource-constrained scheduling
    - Minimum-resource, latency-constrained scheduling

We consider one such heuristic from a family of heuristics called list scheduling that looks at the minimum-latency, resource-constrained scheduling problem

**List Scheduling (LIST_L)**

- Extension of Hu’s algorithm to handle multiple operation types and multiple-cycle execution delays
- Considers minimum-latency, resource-constrained scheduling problem

**List Scheduling (LIST_L)**

- Selection of which operations to include is based on a priority list indicating some sort of urgency measure
  - We will utilize same method of labeling vertices with weights indicating path to sink, choose operations with highest weights
LIST_L Scheduling
Example 1

Step 1

1. Assume all operations take 1 cycle
2. \( s_l = 2 \) multipliers
3. \( s_k = 2 \) ALUs

Step 1

1 = 1

\[ a_1 = 2 \] multipliers
\[ a_2 = 2 \] ALUs

Assume all operations take 1 cycle

\[ \begin{array}{c}
\frac{+}{2} \\
\frac{-}{2}
\end{array} \]
\[ \begin{array}{c}
\frac{\times}{4} \\
\frac{\times}{4}
\end{array} \]
\[ \begin{array}{c}
\frac{\times}{3} \\
\frac{\times}{3}
\end{array} \]
\[ \begin{array}{c}
\frac{\times}{2}
\end{array} \]

TIME 1

1 = 1

Step 2/3

U

Step 2/3

U

T

Step 4

S

S

Step 5

S

S

Step 6

I = I + 1

Step 6

I = I + 1

Step 7

Has \( v_n \) been scheduled yet?

S = \{ v_1 \}

S = \{ v_2 \}

No. Repeat loop.

I = I + 1 = 2

Set vertices in S to start at 1

S = \{ v_11 \}

U = \{ v_11 \}

T = \{ \}

I = 2 + 1 = 3

No. Repeat loop.

Set vertices in S to start at 3

S = \{ v_4 \}

U = \{ v_4 \}

T = \{ \}

I = 3 + 1 = 4

No. Repeat loop.

S = \{ v_7, v_8 \}

U = \{ v_7, v_8 \}

T = \{ \}

S = \{ v_7, v_8 \}

U = \{ v_4 \}

T = \{ \}

Step 4

S = \{ v_1, v_2 \}

S = \{ v_1, v_2 \}

Step 5

S

S

Step 6

I = I + 1

Step 6

I = I + 1

Step 7

Has \( v_n \) been scheduled yet?

S

S

No. Repeat loop.

I = I + 1 = 2

Set vertices in S to start at 2

S

S

No. Repeat loop.

I = I + 1 = 3

Set vertices in S to start at 2

S

S

No. Repeat loop.

I = I + 1 = 4

Set vertices in S to start at 3

S

S

No. Repeat loop.
Step 2/3

\[ U_{l,k} = \text{candidate operations with predecessors finished at } l \]
\[ T_{l,k} = \text{unfinished operations} \]

**LIST L Scheduling Example 1**

\[ l = 4 \]
\[ a_1 = 2 \text{ multipliers} \]
\[ a_2 = 2 \text{ ALUs} \]

Assume all operations take 1 cycle

\[ +/- / 2 \]
\[ */ 2 \]
\[ */ 3 \]
\[ */ 4 \]

Step 4

S = subset set of vertices in U and T such that \( U + T \) is <=a, where labels are maximal

Step 5

Schedule vertices in S to time step I

Step 6

I = I + 1

Step 7

Has \( v_n \) been scheduled yet?

\[ S = \{ \} \]

Multipliers
\[ U = \{ \} \]
\[ T = \{ \} \]

ALUs
\[ U = \{ \} \]
\[ T = \{ \} \]

No. Repeat loop.

**LIST L Scheduling Example 2**

\[ l = 1 \]
\[ a_1 = 3 \text{ multipliers} \]
\[ a_2 = 1 \text{ ALU} \]

Assume all operations take 1 cycle

\[ +/- / 2 \]
\[ */ 2 \]
\[ */ 3 \]
\[ */ 4 \]

Step 4

S = subset set of vertices in U and T such that \( U + T \) is <=a, where labels are maximal

Step 5

Schedule vertices in S to time step I

Step 6

I = I + 1

Step 7

Has \( v_n \) been scheduled yet?

\[ S = \{ \} \]

Multipliers
\[ U = \{ \} \]
\[ T = \{ \} \]

ALUs
\[ U = \{ \} \]
\[ T = \{ \} \]

Set vertices in S to start at 1
LIST_L Scheduling

Example 2

Step 3
U = candidate operations with predecessors finished at l
T = unfinished operations

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Step 5
Schedule vertices in S to time step l

Step 6
I = I + 1

Step 7
Has v_i been scheduled yet?

If yes, go to step 8
If no, set vertices in S to start at l

Step 8
A_o = 3 multipliers
A_n = 1 ALU

Example 2

Step 3
U = candidate operations with predecessors finished at l
T = unfinished operations

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Step 5
Schedule vertices in S to time step l

Step 6
I = I + 1

Step 7
Has v_i been scheduled yet?

If yes, go to step 8
If no, set vertices in S to start at l

Step 8
A_o = 3 multipliers
A_n = 1 ALU

Example 2

Step 3
U = candidate operations with predecessors finished at l
T = unfinished operations

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Step 5
Schedule vertices in S to time step l

Step 6
I = I + 1

Step 7
Has v_i been scheduled yet?

If yes, go to step 8
If no, set vertices in S to start at l

Step 8
A_o = 3 multipliers
A_n = 1 ALU

Example 2

Step 3
U = candidate operations with predecessors finished at l
T = unfinished operations

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Step 5
Schedule vertices in S to time step l

Step 6
I = I + 1

Step 7
Has v_i been scheduled yet?

If yes, go to step 8
If no, set vertices in S to start at l

Step 8
A_o = 3 multipliers
A_n = 1 ALU

Example 2

Step 3
U = candidate operations with predecessors finished at l
T = unfinished operations

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Step 5
Schedule vertices in S to time step l

Step 6
I = I + 1

Step 7
Has v_i been scheduled yet?

If yes, go to step 8
If no, set vertices in S to start at l

Step 8
A_o = 3 multipliers
A_n = 1 ALU
LIST_L Scheduling

Example 2

Step 1 (i = 1)

Step 2/3

U = { } T = { } U = { } T = { } S = { } S = { }

S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

T = { } U = { v_is } Multipliers

A_1 = 3 multipliers A_2 = 1 ALU

Step 4

Step 5

Step 6

Step 7

I = I + 1

Schedule vertices in S to time step I

Has v_i been scheduled yet?

Schedule vertices in S to start at 6

No. Repeat loop.

List Scheduling (LIST_R)

- Considers minimum-resource, latency-constrained scheduling problem

\[
\text{LIST_R}(G(V,E), \lambda) = \begin{cases} 
    a = 1; \\
    \text{Compute the latest possible start times } t^p \text{ by ALAP}(G(V,E), \lambda); \\
    \text{if } t^p < 0 \text{ return } (0); \\
    i = 1; \\
    \text{repeat } \\
    \text{for each resource type } k = 1, 2, ..., n_{res} \{ \\
    \text{Determine candidate operations } U_k; \\
    \text{Compute the slack } s_i = t^p - \lambda_{v_i}; \forall v_i \in U_k; \\
    \text{Schedule the candidate operations with zero slack and update } a_i; \\
    \text{Schedule the candidate operations requiring no additional resources; } \\
    \}
    i = i + 1; \}
    \text{until } (v_i \text{ is scheduled); return } (t, a); \\
\end{cases}
\]

Vector a indicates the number of each type of resource available

Algorithm exits if ALAP detects no feasible solution with dedicated resources

Time step

Operations of type k whose predecessors are completed by time t

Compute slack of all candidates

(ALAP time - current time)

Scheduled any operation with 0 slack to meet timing requirement, add resources if needed

Fill in unused resources by scheduling any available operation

Keep going until we have scheduled the sink node v_n.
LIST_R Scheduling

Example 1

- Assume all operations have unit delay, latency of 4 is required
  - Initialize vector a so all entries have value of 1
  - Compute the latest start times of all nodes by using ALAP()
  - Set time step equal to 1

Schedule candidate operations requiring no additional resources
- no spare multipliers
- S = \{(v_4, v_9)\}

Increment time step:
- i = 2 + 1 = 3
- No. Repeat loop.
LIST_R Scheduling

Example 1

<table>
<thead>
<tr>
<th>Node</th>
<th>Time</th>
<th>Multipliers</th>
<th>ALUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>V_5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>V_10</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>V_11</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Determine candidate operations

U = \{ \Phi \}

Schedule candidate operations with zero slack

S = \{ V_5, V_9 \}

Increment time step

I = 4 + 1 = 5

Schedule candidate operations requiring no additional resources

U = \{ V_n \}

Schedule candidate operations with zero slack and update a

This is the one we will consider

Force-Directed Scheduling (FDS)

- Heuristic scheduling algorithms
  - Consider the unscheduled CDFG under a physics-based spring model
  - Developers are subjected to physical 'forces', both repelling and attracting them to particular time slices
    - Larger the force, the larger the concurrency
  - Goal is to find the optimal placement of vertices into a schedule, when subject to these 'forces'

- Minimum latency under resource-constraint
  - Force directed list scheduling
  - Extension of list scheduling algorithms

- Minimum resource under latency-constraint
  - Force directed scheduling

Force-Directed Scheduling (FDS)

- Force-Directed Scheduling
  - Minimum resource under latency constraint

FDS( G(V,E), \lambda ){
  repeat {
    Compute the time frames;
    Compute the operations and type probabilities;
    Compute the self-forces, predecessor/successor forces and total forces;
    Schedule the operation with least force and update its time-frame;
  } until (all operations scheduled);
  return (I);
}
Force-Directed Scheduling (FDS)

**Time Frames**

- Time frame of an operation is the time interval where it can be scheduled
  - Denoted by \([t_S^i, t_L^i]; i = 0, 1, ..., n\)
  - Earliest and latest start times can be computed by ASAP and ALAP algorithms

  \[ t_S^i = \min \{ t_{ASAP}^j \} \quad \text{and} \quad t_L^i = \min \{ t_{ALAP}^j \} \]

- Width of time frame of an operation is equal to its mobility plus 1

\[ \text{Width} = \text{Mobility} + 1 \]

\[ \text{NOP} \]

\[ \text{V}_0 \quad \text{V}_n \]

\[ \text{TIME 1} \quad \text{TIME 2} \quad \text{TIME 3} \quad \text{TIME 4} \]

**Example 2**

- Time frames for various operations assuming a latency bound of 4
  - Latency bound needed for ALAP scheduling

  \[ \text{Operation } v_1 \quad \text{ASAP time} = 1 \quad \text{ALAP time} = 1 \quad \text{time frame} = [1, 1] \]
  \[ \text{Operation } v_2 \quad \text{ASAP time} = 1 \quad \text{ALAP time} = 2 \quad \text{time frame} = [1, 2] \]
  \[ \text{Operation } v_3 \quad \text{ASAP time} = 1 \quad \text{ALAP time} = 3 \quad \text{time frame} = [1, 3] \]

**Operation Probability**

- Operation Probability is a function
  - Equal to zero outside of the corresponding time frame
  - Equal to reciprocal of the frame width inside the time frame

- Denoted the probability of the operations at time \(i\) by \(p_i(i); i = 0, 1, ..., n\)

- What is the significance?
  - Operations whose time frame is one unit wide are bound to start in one specific time
  - For remaining operations, the larger the width, the lower the probability that the operation is scheduled in any given step inside the corresponding time frame

---

\[ \text{FDS} (G(V,E), \lambda) = \]

repeat {
  Compute the time frames;
  Compute the operations and type probabilities;
  Compute the self-forces, predecessor/successor forces and total forces;
  Schedule the operation with least force and update its time-frame;
} until (all operations scheduled);
return (t);
Example 3

Operation Probability for various operations
- Equal to zero outside of the corresponding time frame
- Equal to reciprocal of the frame width inside the time frame

P = \{v_{1}, \ldots, v_{n}\}

Distribution graph for ALU

Example 4

Distribution graph for ALU
- Sum of probabilities of the operations implemented by a specific resource at any time step of interest

\[ q_{k}(t) = p_{k}(t) = \frac{1}{w} \]

\[ q_{k}(t+1) = \begin{cases} q_{k}(t) & \text{if } q_{k}(t) \leq w \text{ and } q_{k}(t+1) \leq w \\ 0 & \text{otherwise} \end{cases} \]

Distribution graph for Multiplier

Example 5

Distribution graph for Multiplier
- Sum of probabilities of the operations implemented by a specific resource at any time step of interest

\[ q_{k}(t) = \frac{1}{w} \]

\[ q_{k}(t+1) = \begin{cases} q_{k}(t) & \text{if } q_{k}(t) \leq w \text{ and } q_{k}(t+1) \leq w \\ 0 & \text{otherwise} \end{cases} \]

\[ q_{k}(t+1) = 0 \forall k \neq k_{0} \]
Force-Directed Scheduling (FDS)

- Force-Directed Scheduling
  - Minimum resource under latency constraint

FDS( G(V,E), T){
  repeat {
    Compute the time frames;
    Compute the operations and type probabilities;
    Compute the self-forces, predecessor/successor forces and total forces;
    Schedule the operation with least force and update its time-frame;
  } until (all operations scheduled);
  return (t);
}

Example 6

- Calculate Self Force for v6
  - Assignment of v6 to time step 1
  - Assignment of v6 to time step 2

Assuming v6 assigned to time step 1
Self force = 2.8(1-0.5) + 2.3(0-0.5)

Assuming v6 assigned to time step 2
Self force = 2.8(0-0.5) + 2.3(1-0.5)

Want to reduce force (concurrency), time step 2 looks better
How does this impact other operations?
Force-Directed Scheduling (FDS)

**Example 7**

- Calculate Predecessor/Successor Force for \( v_6 \)
  - Assign of \( v_6 \) to time step 1
  - Assign of \( v_6 \) to time step 2

**Assuming \( v_6 \) assigned to time step 2**

- *no predecessor affected*
  - Predecessor force = 0

- *no successor affected*
  - Successor force = 0

**Total force =** \( \text{Self Force} + \text{Predecessor force} + \text{Successor force} \)

\[
\begin{align*}
\text{Predecessor force} &= 0 \\
\text{Successor force} &= 0 \\
\text{Self Force} &= 0.25 + 0 + 0 = 0.25
\end{align*}
\]

**Better choice – want to reduce force in the minimum resource under latency-constraint**

**Assuming \( v_6 \) assigned to time step 1**

\[
\begin{align*}
\text{Predecessor force} &= 0 \\
\text{Successor force} &= 0 \\
\text{Self Force} &= 0.25 + 0 + 0 = 0.25
\end{align*}
\]

**Better choice – want to reduce force in the minimum resource under latency-constraint**
Force-Directed Scheduling (FDS)

- Force-Directed Scheduling
  - Minimum resource under latency constraint

\[
\text{FDS}\left( G(V,E), T \right) = \{
\text{repeat} \{
\text{Compute the time frames;}
\text{Compute the operations and type probabilities;}
\text{Compute the self-forces, predecessor/successor forces and total forces;}
\text{Schedule the operation with least force and update its time-frame;}
\} \text{ until (all operations scheduled);} \text{ return (t); }
\}
\]

At each iteration time frame, probabilities, and forces need to be recalculated

Forces relate to concurrency – we choose lowest force so we can minimize number of resources

Results have shown FDS superior to list scheduling, but run time are long for larger graph (limited usage)

Conclusion

- Considered several types of scheduling algorithms
  - Unconstrained Scheduling – ASAP
  - Latency-Constrained Scheduling – ALAP
  - Resource-Constrained Scheduling – Hu’s Algorithm

- Practical Scheduling problems possibly include multiple-cycle operations with different types
  - Minimum-Latency, Resource-Constrained and Minimum-Resource, Latency-Constrained problems become difficult to solve efficiently
  - Heuristics developed
    - List Scheduling \((\text{LIST}_L)\)
    - List Scheduling \((\text{LIST}_R)\)
    - Force-directed Scheduling
    - Trace Scheduling
    - Percolation Scheduling