ECE 5775
High-Level Digital Design Automation
Fall 2018

Scheduling
Announcements

- Lab 2 released
  - KNN-based digit recognition
  - Due in a week
Outline

▸ More on CFG analysis and SSA
  – Deriving dominance frontier from dominator tree

▸ Unconstrained scheduling
  – ASAP and ALAP

▸ Constrained scheduling
  – Resource constrained scheduling (RCS)
  – Exact formulations with integer linear programming (ILP)
Example: Basic Blocks and CFG

- Can you partition the program into basic blocks and draw the corresponding CFG?

for (i = 0; i < N; i++)
  if (i > 0) A[i-1] = 0;
return;

(1) i = 0
(2) if (i >= N) goto (7)
(3) if (i <= 0) goto (5)
(4) A[i-1] = 0
(5) i++
(6) goto (2)
(7) return

Leader statement is:
(1) the first in the program
(2) any that is the target of a branch
(3) any that immediately follows a branch
Dominator Tree

▶ A node (basic block) \( N \) in CFG may have multiple dominators, but **only one of them will be closest to \( N \)** and be dominated by all other dominators of \( N \)

▶ A dominator tree is a useful way to represent the dominance relation
  
  - The entry node \( s \) is the root
  - **Each node in the tree is the immediate dominator of its children**
    - Each node \( d \) dominates only its descendants in the tree
Example: Dominator Tree

CFG

entry

Dominator Tree
**Algorithm to compute DF set**

For each **convergence point** X in the CFG

For each predecessor, Y, of X in the CFG

Run up to Z=IDOM(X) in the dominator tree, adding X to DF(N) for each N between [Y, Z)
Example: PHI Node Placement

X is defined in B0 and B4 in non-SSA form

Can you identify all the basic blocks where \( \phi \)-nodes need to be inserted for X in the SSA form?
Review: A Typical HLS Flow

High-level Programming Languages (C/C++, OpenCL, SystemC, ...)

Parsing

Transformations

Intermediate Representation (IR)

Allocation

Scheduling

Binding

RTL generation

Control data flow graph (CDFG)

Finite state machines with datapath

if (condition) {
    ...
} else {
    t_1 = a + b;
    t_2 = c * d;
    t_3 = e + f;
    t_4 = t_1 * t_2;
    z = t_4 - t_3;
}

3 cycles
Scheduling in High-Level Synthesis

- Scheduling: a central problem in HLS
  - Introduce clock boundaries to untimed or partially timed input specification
  - Significant impact on QoR
    - Frequency
    - Latency
    - Throughput
    - Area
    - Power
    ...

...
# Scheduling: Untimed to Timed

<table>
<thead>
<tr>
<th></th>
<th>Latency</th>
<th>Area</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untimed</td>
<td>$t_{\text{clk}} \approx 3 \times d_{\text{add}}$</td>
<td>$A_1 = 3 \times A_{\text{add}}$</td>
<td>$t_{\text{clk}} \approx d_{\text{add}} + d_{\text{setup}}$</td>
</tr>
<tr>
<td></td>
<td>$T_1 = 1 / t_{\text{clk}}$</td>
<td></td>
<td>$T_3 = 1 / t_{\text{clk}}$</td>
</tr>
<tr>
<td>Combinational</td>
<td>$T_2 = 1 / (3 \times t_{\text{clk}})$</td>
<td>$A_2 = A_{\text{add}} + 2 \times A_{\text{reg}}$</td>
<td>$A_3 = 3 \times A_{\text{add}} + 6 \times A_{\text{reg}}$</td>
</tr>
<tr>
<td>Sequential</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelined</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Control-Data Flow Graph

- **Untimed**
- **Combinational**
- **Sequential**
- **Pipelined**
Scheduling Input

- Control data flow graph (CDFG)
  - Generated by a compiler front end from high-level description
  - Nodes
    - Operations (and pseudo operations)
  - Directed edges
    - Data edges, control edges, precedence edges

- Without control flow, the basic structure is a data flow graph (DFG)

```
xl = x+dx;
ul = u-3*x*u*dx-3*y*dx
yl = y+u*dx
c = xl<a;
x = xl; u = ul; y = yl;
```
Scheduling Output

- Scheduling: map operations to states
- Each clock cycle corresponds to a state in the FSM
  - Commonly referred to as control step (c-step)

DFG

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>×v1</td>
<td>×v2</td>
<td>×v3</td>
<td>×v6</td>
</tr>
<tr>
<td>×v5</td>
<td>+v9</td>
<td>×v7</td>
<td>×v8</td>
</tr>
<tr>
<td>-v4</td>
<td>+v10</td>
<td>-v11</td>
<td></td>
</tr>
</tbody>
</table>

FSM or State Transition Diagram (STG)

\[ s_1, v1, v2 \]
\[ s_2, v3, v6, v10 \]
\[ s_3, v4, v7, v8 \]
\[ s_4, v5, v9, v11 \]
Unconstrained Scheduling

- Only consideration: dependence

- As soon as possible (ASAP)
  - Schedule an operation to the earliest possible step

- As late as possible (ALAP)
  - Schedule an operation to the earliest possible step, without increasing the total latency
ASAP Schedule

\[ Y = ((a \times b) + c) + (d \times e) - (f + g) \]

**Assumption:** Node delay = Cycle time

The start time for each operation is the least one allowed by the dependencies.

\[
\text{ASAP}(G(V,E)):
V' = \text{Topological\_Sort}(G)
\text{foreach} \ v_i \text{ in } V':
\quad \text{// Primary inputs (PIs) to first cycle}
\quad \text{if } v_i \text{ \in PIs: } t_i = 1
\quad \text{// Assume no chaining & single-cycle operations}
\quad \text{else: } t_i = \max(t_j + 1); \quad (v_j, v_i) \in E
\]
ALAP Schedule

ALAP\((G(V, E), L)\): // L is the latency bound
\[ V' = \text{Reverse\_Topological\_Sort}(G) \]

\textbf{foreach} \( v_i \) in \( V' \):
   // Primary outputs (POs) to last cycle
   \textbf{if} \( v_i \in \text{POs}: \ t_i = L \)
   // Assume no chaining & single-cycle operations
   \textbf{else}: \( t_i = \min(t_j) - 1; // (v_i, v_j) \in E \)

The end time of each operation is the latest one allowed by the dependencies and the latency constraint

\[ Y = ((a*b)+c)+(d*e)-(f+g) \]
Course Selection using ASAP/ALAP?

Computer Engineering Track for ECE and CS majors

Freshman
- ENGRD 2100
- ECE 1210
- CS 111x

Freshman/Sophomore
- ENGRD 2300
- ENGRD 2210

Sophomore/Junior
- ECE 3150
- ECE 3140
- CS 3420
- ECE 4750
- CS 4420
- ECE 4740
- ECE 4760
- ECE 4750
- CS 4420
- ECE 4760

Junior/Senior M.Eng. Ph.D.
- ECE 4740
- ECE 4760
- ECE 4750
- CS 4420
- ECE 5740
- ECE 5760
- ECE 5750
- CS 5420
- ECE 5740
- ECE 5750
- CS 5420

Senior M.Eng. Ph.D.
- ECE 5740
- ECE 5760
- ECE 5750
- CS 5420
- ECE 5745
- ECE 5730
- ECE 5770
- ECE 5775
- CS 5220

Legend:
- VLSI/Circuits
- Architecture
- Systems

Figure credit: Prof. José Martinez, Cornell
Constrained Scheduling

- 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 scheduling
  - SDC-based scheduling
  ...
Linear Programming

- **Linear programming (LP)** solves the problem of maximizing or minimizing a linear objective function subject to linear constraints
  - Efficiently solvable both in theory and in practice

- **Integer linear programming (ILP)**: in addition to linear constraints and objective, the values for the variables have to be integer
  - NP-Hard in general (A special case, 0-1 ILP)
  - Modern ILP solvers can handle problems with nontrivial size

- Enormous number of problems can be expressed in LP or ILP
Canonical Form of ILP

\[
\begin{align*}
\text{maximize} & \quad c_1 x_1 + c_2 x_2 + \ldots + c_n x_n & \quad \text{// objective function} \\
\text{subject to} & \quad a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq b_1 \\
& \quad a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \leq b_2 \\
& \quad \ldots \\
& \quad a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq b_m \\
& \quad x_i \geq 0 \\
& \quad x_i \in \mathbb{Z}
\end{align*}
\]

\textbf{Vector form}

\[
\begin{align*}
\text{maximize} & \quad c^T x & \quad \text{// } c = (c_1, c_2, \ldots, c_n) \\
\text{subject to} & \quad A x \leq b & \quad \text{// } A \text{ is a } m \times n \text{ matrix; } b = (b_1, b_2, \ldots, b_n) \\
& \quad x \geq 0 \\
& \quad x_i \in \mathbb{Z}
\end{align*}
\]
Example: Course Selection Problem

- A student is about to finalize her course selection for the coming semester, given the following information:
  - Minimum credits / semester: 8

<table>
<thead>
<tr>
<th>Course</th>
<th>Schedule</th>
<th>Credits</th>
<th>Est. workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Big data analytics</td>
<td>MW 2:00-3:30pm</td>
<td>3</td>
<td>8 hrs</td>
</tr>
<tr>
<td>2. How to build a start-up</td>
<td>TT 2:00-3:00pm</td>
<td>2</td>
<td>4 hrs</td>
</tr>
<tr>
<td>3. Linear programming</td>
<td>MW 9:00-11:00am</td>
<td>4</td>
<td>10 hrs</td>
</tr>
<tr>
<td>4. Analog circuits</td>
<td>TT 1:00-3:00pm</td>
<td>4</td>
<td>12 hrs</td>
</tr>
</tbody>
</table>

**Question:** Which courses to take to minimize the amount of work?
ILP Formulation for Course Selection

- Define decision variables 
  \( i = 1, 2, 3, 4 \):
  \( x_i = \begin{cases} 
  1 & \text{if course } i \text{ is taken} \\
  0 & \text{if not} 
  \end{cases} \)

- The total expected work hours: 
  \( 8x_1+4x_2+10x_3+12x_4 \)

- The total credits taken: 
  \( 3x_1+2x_2+4x_3+4x_4 \)

- Account for the schedule conflict: 
  \( x_2+x_4 \leq 1 \)

- Complete ILP formulation (in canonical form):
  \[ \text{minimize } 8x_1+4x_2+10x_3+12x_4 \]
  \[ \text{s.t. } \]
  \[ 3x_1+2x_2+4x_3+4x_4 \geq 8 \]
  \[ x_2+x_4 \leq 1 \]
  \[ x_i \in \{0,1\} \]
Resource Constrained Scheduling (RCS)

- When functional units are limited
  - Each functional unit can only perform one operation at each clock cycle
    - e.g., if there are only K adders, no more than K additions can be executed in the same c-step

- A resource-constrained scheduling problem for DFG
  - Given the number of functional units of each type, minimize latency
  - NP-hard problem
ILP Formulation of RCS

- Use binary decision variables
  - $x_{ik} = 1$ if operation $i$ starts at step $k$, otherwise $= 0$
  - $i = 1, \ldots, N$, $N$ is the **total number of operations**
  - $k = 1, \ldots, L$, $L$ is the given **upper bound on latency**

$$t_i = \sum_{k=1}^{L} kx_{ik}$$

$t_i$ indicates the start time of operation $i$
ILP Formulation of RCS: Constraints (1)

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

\[\downarrow\]

\( v_j \) must not start before \( v_i \) completes since \( v_j \) depends on \( v_i \)
Start Time vs. Time(s) of Execution

- $d_i$: latency of operation $i$
  - $d_i = 0$ means a single-cycle operation
  - $d_i > 0$ indicates a multi-cycle operation
- When $d_i = 0$, the following questions are the same:
  - Does operation $i$ start at step $k$
  - Is operation $i$ running at step $k$
- But if $d_i > 0$, the two questions should be formulated as:
  - Does operation $i$ **start** at step $k$
    - Check if $x_{ik}$ is 1
  - Is operation $i$ **running** at step $k$
    - Check if the following hold
      $$\sum_{l=k-d_i}^{k} x_{il} = 1$$
Operation $v_i$ Still Running at Step $k$?

- Is $v_9 \ (d_9 = 2)$ running at step 6?
  
  If and only if $x_{9,6} + x_{9,5} + x_{9,4}$ equals 1

![Diagram](image)

- 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 RCS: Constraints (2)

- Linear constraints:
  - Unique start times: \( \sum_k x_{ik} = 1, \quad i = 1, 2, ..., N \)
  - Dependence must be satisfied (no chaining)
    \[ t_j \geq t_i + d_i + 1 : \forall (v_i, v_j) \in E \Rightarrow \sum_k k \cdot x_{jk} \geq \sum_k k \cdot x_{ik} + d_i + 1 \]
  - 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 of operation \( v_i \) (between 1~n_{res})
\( a_r \) is the number of available resources for resource of type \( r \)
Next Class

- More scheduling algorithms
Acknowledgements

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