Lecture 7

Introduction to Instruction Scheduling

I. Basic Block Scheduling
II. Global Scheduling Concepts

Reading: Chapter 10.3 – 10.4
Scheduling Constraints

- **Data dependences**
  - The operations must generate the same results as the corresponding ones in the original program.

- **Control dependences**
  - All the operations executed in the original program must be executed in the optimized program.

- **Resource constraints**
  - No over-subscription of resources.

Same constraints for instruction level / processor level parallelism
Data Dependence

• **Must maintain order of accesses to potentially same locations**
  – True dependence: write -> read (RAW hazard)
    \[
    a = \ldots \\
    = a
    \]
  – Output dependence: write -> write (WAW hazard)
    \[
    a = \ldots \\
    a = \ldots
    \]
  – Anti-dependence: read -> write (WAR hazard)
    \[
    = a \\
    a = \ldots
    \]

Quiz: What is missing?

• **Data Dependence Graph**
  – **Nodes**: operations
  – **Edges**: \( n_1 \rightarrow n_2 \) if \( n_2 \) is data dependent on \( n_1 \)
    – labeled by the execution length of \( n_1 \)
Analysis on Memory Variables

- **Undecidable in general**
  
  \[
  \text{read } x \\
  \text{read } y \\
  A[x] = \ldots \\
  \ldots = A[y]
  \]

- Two memory accesses can potentially be the same unless proven otherwise

- Classes of analysis:
  - simple: \( \text{base} + \text{offset}_1 = \text{base} + \text{offset}_2 \) ?
  - “data dependence analysis”:
    
    Array accesses whose indices are affine expressions of loop indices
  - interprocedural analysis: \( \text{global} = \text{parameter} \) ?
  - pointer analysis: \( \text{pointer}_1 = \text{pointer}_2 \) ?

- Data dependence analysis is useful for many other purposes
Resource Constraints

- Each instruction type has a resource reservation table

<table>
<thead>
<tr>
<th>Functional units</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld</td>
</tr>
<tr>
<td>st</td>
</tr>
<tr>
<td>alu</td>
</tr>
<tr>
<td>fmpy</td>
</tr>
<tr>
<td>fadd</td>
</tr>
<tr>
<td>br</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

- Pipelined functional units: occupy only one slot
- Non-pipelined functional units: multiple time slots
- Instructions may use more than one resource
- A resource type may have multiple units
- Limited instruction issue slots
  - may also be managed like a resource
Example of a Machine Model

- Each machine cycle can execute 2 operations
- 1 ALU operation or branch operation
  \[ \text{Op} \ dst, \ src1, \ src2 \] executes in 1 clock
- 1 load or store operation
  \[ \text{LD} \ dst, \ addr \] result is available in 2 clocks pipelined: can issue LD next clock
  \[ \text{ST} \ src, \ addr \] executes in 1 clock cycle
  - can issue LD/ST to any location in the next clock
  - a write buffer is often used to hold the result
Basic Block Scheduling Example

LD R2 <- 0(R1)
ST 4(R1) <- R2
LD R3 <- 8(R1)
ADD R3 <- R3,R4
ADD R3 <- R3,R2
ST 12(R1) <- R3
ST 0(R7) <- R7

Quiz: Shown are the register data dependence constraints. What are the memory data dependence constraints?
With Resource Constraints

- **NP-complete in general → Heuristics time!**
- **List Scheduling:**

  \[ \text{READY} = \text{nodes with 0 predecessors} \]

  Loop until READY is empty {

  Let \( n \) be the node in READY with **highest priority**

  Schedule \( n \) in the **earliest** slot 
  that satisfies precedence + resource constraints

  Update predecessor count of \( n \)'s successor nodes 
  Update READY

  }


List Scheduling for Basic Blocks

- **Scope:** DAGs
  - Schedules operations in topological order
  - Never backtracks

- **Variations:**
  - Priority function for node \( n \)
    - critical path: max clocks from \( n \) to any node
    - resource requirements
    - source order

Quiz: Why use the simple list scheduling algorithm for basic blocks?

Quiz: How many operations are in a basic block?

Quiz: How much parallelism is in a basic block?
II. Introduction to Global Scheduling

Assume each clock can execute 2 operations of any kind & no aliases

Quiz: Is there parallelism in this program? How do you schedule it?
Result of Code Scheduling

LD R6 ← 0(R1)
nop
BEQZ R6, L

LD R7 ← 0(R2)
nop
ST 0(R3) ← R7

LD R8 ← 0(R4)
nop
ADD R8 ← R8, R8
ST 0(R5) ← R8

L:
B1

B2

B3

B3'

LD R6 ← 0(R1) ; LD R8 ← 0(R4)
LD R7 ← 0(R2)
ADD R8 ← R8, R8 ; BEQZ R6, L

L:
ST 0(R5) ← R8

ST 0(R5) ← R8 ; ST 0(R3) ← R7
Lessons from the Example

• Basic blocks are small, lots of dependences

• Global scheduling (across basic blocks) is necessary
  – Lots of dependences esp. with memory operations
    (esp. due to aliases)

• Static schedulers can look ahead & prioritize over dynamic schedulers.
Control Dependence Constraints

**Control equivalence:**
- Two operations $o_1$ and $o_2$ are *control equivalent* if $o_1$ is executed if and only if $o_2$ is executed.

**Control dependence:**
- An op $o_2$ is *control dependent* on op $o_1$ if the execution of $o_2$ depends on the outcome of $o_1$.

**Speculation:**
- An operation $o$ is *speculatively* executed if it is executed before all the operations it depends on (control-wise) have been executed.
- Requirement:
  - Raises no exception,
  - Satisfies data dependences

**Quiz:** what are the differences between static and dynamic schedulers?
Summary

• **List scheduling**
  – Greedy algorithm with no backtracking
  – Topological sort with a priority function to choose among candidates

• **Global scheduling**
  – Limited opportunity of code motion within a single basic block
  – Concepts of control dependence, control equivalence, and speculative execution