Lecture 7
Instruction Scheduling

I. 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.
Data Dependence

- Must maintain order of accesses to potentially same locations

  - True dependence: write -> read (RAW hazard)
    \[
    \begin{align*}
    a &= \ldots \\
    &= a
    \end{align*}
    \]

  - Output dependence: write -> write (WAW hazard)
    \[
    \begin{align*}
    a &= \ldots \\
    a &= \ldots
    \end{align*}
    \]

  - Anti-dependence: read -> write (WAR hazard)
    \[
    \begin{align*}
    &= a \\
    a &=
    \]

- 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
  
  \[
  \begin{align*}
  \text{read } x \\
  \text{read } y \\
  A[x] &= \ldots \\
  \ldots &= A[y]
  \end{align*}
  \]

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

- Classes of analysis
  
  - simple: \( \text{base+offset1 = base+offset2?} \)
  
  - “data dependence analysis”: Array accesses whose indices are affine expressions of loop indices
    \[
    \]

  - interprocedural analysis: \( \text{global = parameter?} \)

  - pointer analysis: \( \text{pointer1 = pointer2?} \)

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

- Each instruction type has a resource reservation table

```
| Time | ld | st | alu | fmpy | fadd | br | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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
- Multiple units of same resource
- Limited instruction issue slots may also be managed like a resource

Example of a Machine Model

- Each instruction can execute 2 operations

  - 1 ALU operation or branch operation
    \[\text{Op} \ dst, src1, src2 \text{ executes in 1 clock}\]

  - 1 load or store operation
    \[\text{LD} \ dst, \ addr \text{ result is available in 2 clocks}\]
    \[\text{ST} \ src, \ addr \text{ pipelined: can issue LD next clock}\]
    \[\text{ST} \ src, \ addr \text{ executes in 1 clock cycle}\]
Basic Block Scheduling

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

  }

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
List Scheduling

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

- **Variations**
  - Priority function for node $n$
    - delay: max delay slots from $n$ to any node
    - critical path: max clocks from $n$ to any node
    - resource requirements
    - source order

II. Introduction to Global Scheduling

```
if (a=0) goto L

L:
c = b

e = d + d

LD R6,0(R1)
BEQZ R6,L

LD R7,0(R2)
ST 0(R3),R7

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

L:
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

B1

B2

B3
```
**Result of Code Scheduling**

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

**Terminology**

**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 $o_1$ is speculatively executed if it is executed before all the operations it control-dependent upon have been executed.

- No exception, satisfy data dependences
**Code Motions**

Goal: Shorten execution time probabilistically

Moving instructions up
- Move instruction to a cut set (from entry)
- Speculation: even when not anticipated.

Moving instructions down
- Move instruction to a cut set (from exit)
- May execute extra instruction
- Can duplicate code

**A Note on Data Dependences**

```
 a = 0    a = 1
  src  
```

```
General-Purpose Applications

- Lots of data dependences
- Key performance factor: memory latencies
- Move memory fetches up
  - Speculative memory fetches can be expensive
- Control-intensive: get execution profile
  - Static estimation
    - Innermost loops are frequently executed: back edges are likely to be taken
    - Edges that branch to exit and exception routines are not likely to be taken
  - Dynamic profiling
    - Instrument code and measure using representative data

A Basic Global Scheduling Algorithm

- Schedule innermost loops first
- Only upward code motion
- No creation of copies
- Only one level of speculation
Program Representation

- A region in a control flow graph is
  - a set of basic blocks and all the edges connecting these blocks,
  - such that control from outside the region must enter through a single entry block.

- A function is represented as a hierarchy of regions
  - The whole control flow graph is a region
  - Each natural loop in the flow graph is a region
  - Natural loops are hierarchically nested

- Schedule regions from inner to outer
  - treat inner loop as a black box unit, can schedule around it but not into it
  - ignore all the loop back edges --> get an acyclic graph

Algorithm

Compute data dependences;
For each region from inner to outer {
  For each basic block B in prioritized topological order {
    CandBlocks = ControlEquiv(B) ∪ Dominated-Successors(ControlEquiv(B));
    CandInsts = ready operations in CandBlocks;
    For (t = 0, 1, ... until all operations from B are scheduled {
      For (n in CandInst in priority order) {
        if (n has no resource conflicts at time t) {
          S(n) = < B, t >
          Update resource commitments
          Update data dependences
        }
      }
    }
    Update CandInsts;
  }}

- Priority functions
  - Non-speculative before speculative
Extensions

• Prepass before scheduling: loop unrolling

• Especially important to move operation up loop back edges

Summary

• List scheduling

• Global scheduling
  • Legal code motions
  • Heuristics