7.8 Trace Scheduling
- Bookkeeping

Graph
- Nodes
  - Operations, "instructions", micro-ops"
- Edges
  - Dependencies
    - Split/join
    - Leave/enter the trace

Schedule
- Task (records the decision)

<table>
<thead>
<tr>
<th>Slot 0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>k</td>
</tr>
</tbody>
</table>

Operation(s)
Instruction
7.8.2 Joins

join node: node in the trace that is a target of a join edge.

Two issues must be dealt with by bookkeeping:

1) What should happen if a node that is (in the trace) below the join node ends up in the schedule above the join node?

2) What should be the target of the join edge (join node moved, or instruction relative to join node)?
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>E</td>
</tr>
</tbody>
</table>

Trace

Schedule

1 moved above the join node

Another schedule (assuming no data dependencies are violated)
If an operation moved from below the join node to above the join point on the schedule, we must insert a copy of that node in the off-trace-path.

- copy: compensation cycle

(assume: slot #3 is the target of the join edge after scheduling)
where should the off-trace path (1|join the schedule?  

We cannot pick the dot that holds the first instruction from below the join

\[ \text{EB} \]  (in this example: D)  
we do not know the effect of B!  

The earliest place to join the schedule is the dot T such that all nodes that appear above the join in the graph appear in dots 0...\(T-D\) of the schedule.
E is in slot 4: B is the last node from "above C in graph" that is scheduled into slot 3.
7.8.3 Split

Split node: node on the trace where at least one edge leaves the trace.

2 issues for bookkeeping:
1) node moves from below the split to above the split
2) node moves from above the split to below the split
1) we cannot allow arbitrary moves from below-the-split-node to above-the-split-node.

Nodes can be moved iff they produce no side effects visible in the off-trace-path.

- instrs that modify dead registers (\rightarrow in the off-trace-path)
- instrs that delay reporting exceptions
- prefetch instrs / speculative load (helps with caches, must create a prefetch variant of nodes that are moved)
2) BookKeeper must insert a copy of the moved instruction into the off-trace-path compensation code.
One useful metric to assess the usefulness of trace scheduling is what percentage of the execution time is spent on the first trace picked by the compiler.

- Dense matrix multiplication: 92%
- FFT: 85%
- Chemistry simulator: 42%
- Lisp interpreter: 10%
- gcc: 8%
7.9. Software pipelining

Pipelining: overlap stages

Example machine:

\[
\begin{align*}
\text{1 load } f & \text{ in parallel} \\
\text{1 store } f & \text{ in parallel} \\
\text{1 add } & \\
\text{1 mut } & \\
\text{1 branch } & \\
\end{align*}
\]

all operations have 2 cycles latency

for \( i = 0; i < n \) do

\[
x[i] = a[i] \times c + x[i];
\]

\}
innermost loop - computations

\[ \text{loop} \]

\[ \text{Id} \times [i,j] \rightarrow R_1, \]
\[ \text{Id} a[i] \rightarrow R_2, \]
\[ \text{mul} \ R_0, R_2 \rightarrow R_3, \]
\[ \text{add} \ R_3, R_1 \rightarrow R_4, \]
\[ \text{st} \ R_4 \rightarrow x[i,j] \]

branch

Id

\[ \text{mul} \]
\[ \text{add} \]
\[ \text{st} \]
\[ \text{br} \]
One way to improve performance: unroll 4x unroll

\[
\begin{align*}
1d & x
1d^2 & x
1d^3 & x
1d & x
1d^2 & x
1d^3 & x
1d & x
1d^2 & x
& \text{mul}^1 \\
& \ldots \\
& \text{add}^1 \\
& \text{mul}^2 \\
& \text{mul}^3 \\
\end{align*}
\]

Some overlap
problem with unrolling

not all resources are used
e.g. example: only loads

loop

Start-up

Keep all functional units busy

store results

wind down

ld/st and mul/adc
Idea: take sequence of operations and overlap loop iterations directly

- Id x
- Id a
- mul
- add
- st
Try to find a schedule that allows overlap.
- Many algorithms (take the plan to schedule into account)
- Dependencies between operations

Registers are an issue
- Use different registers for second, third...
- Iteration
- Pipelined registers supported by hardware

Conditional start a potential show stopper

Use software pipelining for (simple) inner-most loop
Use trace scheduling for the next ke traces
- Use simple compiler for the rest of the program.