Instruction Scheduling

Last time
- Instruction scheduling using list scheduling

Today
- Improvements on list scheduling
  - Register renaming
  - Unrolling
  - Software pipelining

Improving Instruction Scheduling

Techniques
- Register renaming
- Scheduling loads
- Loop unrolling
- Software pipelining
- Predication and speculation

Deal with data hazards
Deal with control hazards

Register Renaming

Idea
- Reduce false data dependences by reducing register reuse
- Give the instruction scheduler greater freedom

Example

```
add $r1, $r2, 1      add $r1, $r2, 1
st $r1, [$fp+52]     st $r1, [$fp+52]
mul $r1, $r3, 2      mul $r11, $r3, 2
st $r1, [$fp+40]     st $r11, [$fp+40]
```

Scheduling Loads

Reality
- Loads can take many cycles (slow caches, cache misses)
- Many cycles may be wasted

Most modern architectures provide non-blocking (delayed) loads
- Loads never stall
- Instead, the use of a register stalls if the value is not yet available
- Scheduler should try to place loads well before the use of target register
Scheduling Loads (cont)

Hiding latency
- Place independent instructions behind loads

load r1, (r3)
load r2, (r1)
add r4
load r1, (r3)
add r4
load r2, (r1)

- How many instructions should we insert?
  - Depends on latency
  - Difference between cache miss and cache hits are growing
  - If we underestimate latency: Stall waiting for the load
  - If we overestimate latency: Hold register longer than necessary
  - Wasted parallelism

Loop Unrolling

Idea
- Replicate body of loop and iterate fewer times
- Reduces loop overhead (test and branch)
- Creates larger loop body ⇒ more scheduling freedom

Example
L: ldf, f0, [rl]
  fadds, f2, f0, f1
  stf, [rl], f2
  sub, r1, 4, r1
  cmp, r1, 0
  bg, L
  nop

Cycles per iteration: 9

Loop Unrolling Example

Sample loop
L: ldf, f0, [rl]
  fadds, f2, f0, f1
  ldf, f10, [rl-4]
  fadds, f12, f10, f1
  stf, [rl], f2
  stf, [rl-4], f12
  sub, r1, 8, r1
  cmp, r1, 0
  bg, L
  nop

The larger window lets us hide some of the latency of the fadds instruction

Loop Unrolling Summary

Benefit
- Loop unrolling allows us to schedule code across iteration boundaries, providing more scheduling freedom

Issues
- How much unrolling should we do?
  - Try various unrolling factors and see which provides the best schedule?
  - Unroll as much as possible within a code expansion budget?
  - An alternative: Software pipelining
Software Pipelining

Basic idea

- **Software pipelining** is a systematic approach to scheduling across iteration boundaries without doing loop unrolling.
- Try to move the long latency instructions to previous iterations of the loop.
- Use independent instructions to hide their latency.

- Three parts of a software pipeline
  - **Kernel**: Steady state execution of the pipeline
  - **Prologue**: Code to fill the pipeline
  - **Epilogue**: Code to empty the pipeline

Visualizing Software Pipelining

Software Pipelining versus Loop Unrolling

Example assumes infinite functional units and single-cycle latency.
SW Pipelining (Step 2: “Unroll”, Satisfy Latencies, Find Pattern)

This pattern does not work!!

SW Pipelining (Step 3: Satisfy register constraints)

SW Pipelining (Step 1: Construct DAG and Assign Registers)

Example assumes pg. 739 machine.

This pattern does not work!!

SW Pipelining and Loop Unrolling Summary

Unrolling removes branching overhead and helps tolerate data dependence latency.

SW pipelining maintains max parallelism in steady state through continuous tolerance of data dependence latency.

Both work best with loops that are parallel, getting ILP by taking instructions from different iterations.

Software Pipelining

Complications

- What if there is control flow within the loop?
  - Use control-flow profiles to identify most frequent path through the loop
  - Optimize for the most frequent path
- How do we identify the most frequent path?
  - Profiling

Concepts

Improving instruction scheduling

- Register renaming
- Scheduling loads
- Loop unrolling

Instruction scheduling across basic blocks

- Software pipelining
Next Time

Lecture
- Data-flow analysis