Instruction Scheduling

Motivation

Instruction latency
(pipeline)

several cycles to complete instruction

Instruction pipeline (VLIW, superscalar)

several cycles to complete instruction

Instruction latency

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Sources of latency (hazards)

data - operands depend on previous execution

limited hardware resources

control - targets of conditional branches

Approach

schedule after register allocation (postpass)

Schedule after register allocation

Schedule

Instruction Scheduling

Scope

Several cycles to complete instruction

Issue

Source of latency (hazards)

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

control - targets of conditional branches

limited hardware resources

Issue

reorder instructions to reduce execution time

process cooperation

insert NOPs

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time

interactions with optimizations

operate efficiently

preserve correctness

Dynamic schedule - pipeline stalls

schedule instruction to reduce execution time
Register renaming eliminates anti and output deps

```plaintext
Sample code:

1. a = b
2. b = a
3. a = a
4. a = a
5. a = a
6. a = a
```

Another dependence (does not constrain ordering)

```plaintext
(\text{read-after-read or RAW})
da reads a location that b later reads
```

```plaintext
(\text{write-after-write or WAW})
a writes a location that b later writes
```

Example:

```plaintext
\text{Register renaming eliminates anti and output deps}
```

```plaintext
\text{Another dependence (does not constrain ordering)}
```

```plaintext
\text{Data Dependences}
```

```plaintext
\text{Precedence Graph}
```

```plaintext
\text{Construction}
```

```plaintext
\text{Nodes} 
\text{Instructions} 
\text{Dependences}
```

```plaintext
\text{Edges} 
\text{Dependences} 
\text{Precedence Graph}
```

```plaintext
\text{Construction}
```

```plaintext
\text{Example}
```

```plaintext
\text{Example}
```

```plaintext
= a = a
```

```plaintext
= a = a
```

```plaintext
= a = a
```

```plaintext
= a = a
```

```plaintext
= a = a
```

```plaintext
\text{time or flow dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```

```plaintext
\text{another dependence}
```
Try multiple schedules (take best result)

use multiple heuristics (help break ties)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Algorithm

List Scheduling

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)

Two flavors of List Scheduling

Forward List Scheduling

start with available ops

Backward List Scheduling

start with no successors

Probe forward as scheduling proceeds

Heuristics to prioritize instructions to candidates

4. highest latency (more overlap)
3. most immediate successors (create candidates)
2. highest latency (more overlap)
1. ready to execute (no stalls)

Heuristics used to prioritize candidates

NP-hard for straight-line code

how to choose between ready instructions?

Problems

Heuristics

Schedule heuristics

1. try multiple schedules (take best result)

2. longest weighted path to root (critical path)

1. most descendants (create more candidates)

1. highest latency (more overlap)
0. ready to execute (no stalls)
### Schedule Example

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Candidates**
- **Forward Schedule**
- **Backward Schedule**

**Example**

- Load 1, r2, r1
- Load 3, r3, r2
- Load 4, r4, A
- Load 6, r6, r5

**Overview**

- Code latency for load
- Code latency for multiply
- Code latency for add

**Trace Scheduling**

- A trace is a path through code
  - Schedule instructions for trace
  - Candidate trace representing most likely path
  - Examine branch probabilities
  - Repeat as necessary

**Result**

- Better instruction schedule along trace
- Less efficient schedule off trace
- Increase in code size (from repair code)
- Better instruction schedule along code

---

**Machine Code**

- Trace scheduling
- Code latency
- Code size

---

**Lecture 1.9, Page 8**
Loops Unrolling

**Approach**

- Instruction cache overflow
- Increased compilation time
- Pipeline bubbling every iteration
- Choosing degree of unrolling

**Problems**

- only move code which is dead or trace may perform unnecessary work

**Example**

```c
    store
    store
    store
    store
    load
    load
    load
    // do i=1,N,3
```

- more candidates for scheduling
- create multiple copies of loop

**Exampl**

```c
    code3
    code2
    (\ldots)
```

- increased compilation time
- increased instruction cache overflow

**Example**

```c
    code3
    code2
    (\ldots)
    code1
    code1
```

- only move code which is dead of trace
- insert copy in b in original order

**Example**

```c
    code3
    code2
    (\ldots)
    code1
    code1
```

- create multiple copies of loop
- create basic block at branch target

**Trace Scheduling Repair Code**
Software Pipelining

Approach

Overlap iterations of loop

Select schedule for loop body

overlap iterations of loop

Example

\[
\begin{align*}
\text{store} & \quad 7 \\
\text{store} & \quad 6 \\
\text{store} & \quad 5 \\
4 \text{ I: store mult load (cmp I)} & \\
3 \quad \text{mult load} & \\
2 \quad \text{load} & \\
1 & \\
T & = T \\
1 & = 2 \\
1 & = 3 \\
1 & = 4
\end{align*}
\]

Properties

- steady state within body of loop
- pipeline overlap code for pipeline

Properties

- steady state within body of loop
- pipeline overlap code for pipeline

Branch Prediction

Will a conditional branch be taken?

Branch Prediction

- catches loop back edges
- predicts conditional branch taken

Simple (target) heuristic

- perfect prediction = 30-100% correct
- single prediction for all executions
- no run-time information

Static branch prediction

- compile-time heuristics
- profile (feedback to compiler)
- history plus branch correlation
- hardware history (branch bit)

Prediction approaches

- execution penalty for incorrect guess
- affects instruction scheduling

Will a conditional branch be taken?
Branch Prediction

Loop Branch Heuristic

• Predict branch leading to loop header

Code Heuristic

• Predict branch not leading to return

• Predict branch not leading to call

Guard Heuristic

• Predict branch leading to guarded variable

Store Heuristic

• Predict branch leading to store of variable

• Predict branch leading to store of variable

Op Code Heuristic

• Predict branch not leading to non-exit edge

• And forward, back, exit edges

Branch Heuristic

Predict branch not leading to NTL, pointers differ

Point Branch Heuristic

Predict branch leading to NTL, pointers differ