Loop Optimizations

Instruction Scheduling
Outline

• Scheduling for loops
• Loop unrolling
• Software pipelining
• Interaction with register allocation
• Hardware vs. Compiler
• Induction Variable Recognition
• loop invariant code motion
Loop Example

• Machine
  – One load/store unit
    • load 2 cycles
    • store 2 cycles
  – Two arithmetic units
    • add 2 cycles
    • branch 2 cycles
    • multiply 3 cycles
  – Both units are pipelined (initiate one op each cycle)

• Source Code
  \[
  \text{for } i = 1 \text{ to } N \\
  \text{A}[i] = \text{A}[i] \times b
  \]
Loop Example

- **Source Code**
  
  ```
  for i = 1 to N
  ```

- **Assembly Code**
  
  ```
  loop:
      mov (%rdi,%rax), %r10
      imul %r11, %r10
      mov %r10, (%rdi,%rax) 
      sub $4, %rax
      jz loop
  ```
Loop Example

- Assembly Code

```assembly
loop:
    mov (%rdi,%rax), %r10
    imul %r11, %r10
    mov %r10, (%rdi,%rax)
    sub $4, %rax
    jz loop
```

Diagram:
- `mov`: d=7
- `imul`: d=5
- `mov`: d=2
- `sub`: d=2
- `jz`: d=0
Loop Example

• **Assembly Code**

```assembly
loop:
    mov (%rdi,%rax), %r10
    imul %r11, %r10
    mov %r10, (%rdi,%rax)
    sub $4, %rax
    jz loop
```

• **Schedule (9 cycles per iteration)**

<table>
<thead>
<tr>
<th>mov</th>
<th>mov</th>
<th>mov</th>
<th>mov</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov</td>
<td>mov</td>
<td>mov</td>
</tr>
<tr>
<td>imul</td>
<td>imul</td>
<td>bge</td>
<td>bge</td>
</tr>
<tr>
<td>imul</td>
<td>imul</td>
<td>bge</td>
<td>bge</td>
</tr>
<tr>
<td>imul</td>
<td>imul</td>
<td>bge</td>
<td>bge</td>
</tr>
<tr>
<td>sub</td>
<td>sub</td>
<td>sub</td>
<td>sub</td>
</tr>
</tbody>
</table>
Outline

• Scheduling for loops
• **Loop unrolling**
• Software pipelining
• Interaction with register allocation
• Hardware vs. Compiler
• Induction Variable Recognition
• loop invariant code motion
Loop Unrolling

• Unroll the loop body few times
• Pros:
  – Create a much larger basic block for the body
  – Eliminate few loop bounds checks
• Cons:
  – Much larger program
  – Setup code (# of iterations < unroll factor)
  – beginning and end of the schedule can still have unused slots
Loop Example

loop:
    mov   (%rdi,%rax), %r10
    imul  %r11, %r10
    mov   %r10, (%rdi,%rax)
    sub   $4, %rax
    jz    loop
Loop Example

```
loop:    mov (%rdi,%rax), %r10
        imul %r11, %r10
        mov %r10, (%rdi,%rax)
        sub $4, %rax
        mov (%rdi,%rax), %r10
        imul %r11, %r10
        mov %r10, (%rdi,%rax)
        sub $4, %rax
        jz loop
```
Loop Example

loop:
  mov  (%rdi,%rax), %r10
  imul %r11, %r10
  mov  %r10, (%rdi,%rax)
  sub  $4, %rax
  mov  (%rdi,%rax), %r10
  imul %r11, %r10
  mov  %r10, (%rdi,%rax)
  sub  $4, %rax
  jz   $0, %rax

• Schedule (8 cycles per iteration)
Loop Unrolling

• Rename registers
  – Use different registers in different iterations
Loop Example

loop:
  mov (%rdi, %rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi, %rax)
  sub $4, %rax
  mov (%rdi, %rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi, %rax)
  sub $4, %rax
  jz loop
Loop Example

```
loop:
    mov (%rdi,%rax), %r10
    imul %r11, %r10
    mov %r10, (%rdi,%rax)
    sub $4, %rax
    mov (%rdi,%rax), %rcx
    imul %r11, %rcx
    mov %rcx, (%rdi,%rax)
    sub $4, %rax
    jz loop
```
Loop Unrolling

• Rename registers
  – Use different registers in different iterations

• Eliminate unnecessary dependencies
  – again, use more registers to eliminate true, anti and output dependencies
  – eliminate dependent-chains of calculations when possible
Loop Example

```assembly
loop:
    mov  (%rdi,%rax), %r10
    imul %r11, %r10
    mov  %r10, (%rdi,%rax)
    sub  $4, %rax
    mov  (%rdi,%rax), %rcx
    imul %r11, %rcx
    mov  %rcx, (%rdi,%rax)
    sub  $4, %rax
    jz   loop
```
Loop Example

loop:
  mov (%rdi,%rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi,%rax)
  sub $8, %rax
  mov (%rdi,%rbx), %rcx
  imul %r11, %rcx
  mov %rcx, (%rdi,%rbx)
  sub $8, %rbx
  jz loop
Loop Example

loop:

mov (%rdi,%rax), %r10
imul %r11, %r10
mov %r10, (%rdi,%rax)
sub $8, %rax
mov (%rdi,%rbx), %rcx
imul %r11, %rcx
mov %rcx, (%rdi,%rbx)
sub $8, %rbx
jz loop

- Schedule (4.5 cycles per iteration)
Outline

• Scheduling for loops
• Loop unrolling
• **Software pipelining**
• Interaction with register allocation
• Hardware vs. Compiler
• loop invariant code motion
• Induction Variable Recognition
Software Pipelining

- Try to overlap multiple iterations so that the slots will be filled
- Find the steady-state window so that:
  - all the instructions of the loop body is executed
  - but from different iterations
## Loop Example

### Assembly Code

```assembly
loop:
    mov (%rdi,%rax), %r10
    imul %r11, %r10
    mov %r10, (%rdi,%rax)
    sub $4, %rax
    jz loop
```

### Schedule

<table>
<thead>
<tr>
<th>mov</th>
<th>mov</th>
<th>mov</th>
<th>mov</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov</td>
<td>mov</td>
<td>mov</td>
</tr>
<tr>
<td>mul</td>
<td>mul</td>
<td>mul</td>
<td>mul</td>
</tr>
<tr>
<td>mul</td>
<td>mul</td>
<td>mul</td>
<td>mul</td>
</tr>
</tbody>
</table>

| sub | sub | sub | sub |
Loop Example

• **Assembly Code**

  ```assembly
  loop:
  mov (%rdi,%rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi,%rax)
  sub $4, %rax
  jz loop
  ```

• **Schedule**

```
<table>
<thead>
<tr>
<th>mov</th>
<th>mov1</th>
<th>mov</th>
<th>mov</th>
<th>mov1</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov1</td>
<td>mov</td>
<td>mov</td>
<td>mov1</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>jz</td>
<td>jz</td>
<td>jz1</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>jz</td>
<td>jz</td>
<td>jz1</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>jz</td>
<td>jz</td>
<td>jz1</td>
</tr>
<tr>
<td>sub</td>
<td>sub1</td>
<td>sub</td>
<td>sub</td>
<td>sub1</td>
</tr>
</tbody>
</table>
```
Loop Example

• Assembly Code

<table>
<thead>
<tr>
<th>mov</th>
<th>mov1</th>
<th>mov2</th>
<th>mov</th>
<th>mov1</th>
<th>mov2</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov1</td>
<td>mov2</td>
<td>mov</td>
<td>mov1</td>
<td>mov2</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>jz1</td>
<td>jz2</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>jz1</td>
<td>jz2</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>jz1</td>
<td>jz2</td>
</tr>
<tr>
<td>mul</td>
<td>sub</td>
<td>sub1</td>
<td>sub2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>sub1</td>
<td>sub2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  loop:
  mov (%rdi,%rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi,%rax)
  sub $4, %rax
  jz loop
Loop Example

- **Assembly Code**

  ```assembly
  loop:
  mov (%rdi,%rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi,%rax)
  sub $4, %rax
  jz loop
  ```

- **Schedule**

<table>
<thead>
<tr>
<th>mov</th>
<th>mov1</th>
<th>mov2</th>
<th>mov</th>
<th>mov3</th>
<th>mov1</th>
<th>mov2</th>
<th>mov3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov1</td>
<td>mov2</td>
<td>mov</td>
<td>mov3</td>
<td>mov1</td>
<td>mov2</td>
<td>mov3</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>jz2</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>jz2</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>sub</td>
<td>sub1</td>
<td>sub2</td>
<td>sub3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Loop Example

### Assembly Code

```assembly
loop:
    mov (%rdi,%rax), %r10
    imul %r11, %r10
    mov %r10, (%rdi,%rax)
    sub $4, %rax
    jz loop
```

### Schedule

<table>
<thead>
<tr>
<th>mov</th>
<th>mov1</th>
<th>mov2</th>
<th>mov</th>
<th>mov3</th>
<th>mov1</th>
<th>mov4</th>
<th>mov2</th>
<th>mov3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov1</td>
<td>mov2</td>
<td>mov</td>
<td>mov3</td>
<td>mov1</td>
<td>mov4</td>
<td>mov2</td>
<td>mov3</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>sub1</td>
<td>sub2</td>
<td>sub3</td>
<td>sub4</td>
<td>sub5</td>
<td>sub6</td>
<td>sub7</td>
<td>sub8</td>
</tr>
</tbody>
</table>
Loop Example

- **Assembly Code**

  ```assembly
  loop:
  mov (%rdi,%rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi,%rax)
  sub $4, %rax
  jz loop
  ```

- **Schedule**

<table>
<thead>
<tr>
<th>mov</th>
<th>mov1</th>
<th>mov2</th>
<th>mov</th>
<th>mov3</th>
<th>mov1</th>
<th>mov4</th>
<th>mov2</th>
<th>mov5</th>
<th>mov3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov2</td>
<td>mov1</td>
<td>mov</td>
<td>mov2</td>
<td>mov3</td>
<td>mov1</td>
<td>mov4</td>
<td>mov2</td>
<td>mov5</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul5</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul4</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul4</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul4</td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>sub1</td>
<td>sub2</td>
<td>sub3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Loop Example

• Assembly Code
  
  ```assembly
  loop:
  mov (%rdi,%rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi,%rax)
  sub $4, %rax
  jz loop
  ```

• Schedule

<table>
<thead>
<tr>
<th>mov</th>
<th>mov1</th>
<th>mov2</th>
<th>mov</th>
<th>mov3</th>
<th>mov1</th>
<th>mov4</th>
<th>mov2</th>
<th>mov5</th>
<th>mov3</th>
<th>mov6</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov1</td>
<td>mov2</td>
<td>mov</td>
<td>mov3</td>
<td>mov1</td>
<td>mov4</td>
<td>mov2</td>
<td>mov5</td>
<td>mov3</td>
<td>mov6</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>sub1</td>
<td>sub2</td>
<td>sub</td>
<td>sub3</td>
<td>sub2</td>
<td>sub3</td>
<td>sub</td>
<td>sub3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Loop Example

• Assembly Code

```
loop:
    mov (%rdi,%rax), %r10
    imul %r11, %r10
    mov %r10, (%rdi,%rax)
    sub $4, %rax
    jz loop
```

• Schedule

<table>
<thead>
<tr>
<th>mov</th>
<th>mov1</th>
<th>mov2</th>
<th>mov</th>
<th>mov3</th>
<th>mov1</th>
<th>mov4</th>
<th>mov2</th>
<th>mov5</th>
<th>mov3</th>
<th>mov6</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov1</td>
<td>mov2</td>
<td>mov</td>
<td>mov3</td>
<td>mov1</td>
<td>mov1</td>
<td>mov4</td>
<td>mov2</td>
<td>mov3</td>
<td>mov3</td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td>mul5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>mul1</td>
<td>mul2</td>
<td>jz</td>
<td>mul3</td>
<td>jz1</td>
<td>mul4</td>
<td>jz2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>sub1</td>
<td>sub2</td>
<td>sub</td>
<td>sub2</td>
<td>sub3</td>
<td>sub3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Loop Example

• **Assembly Code**

```assembly
    loop:
        mov (%rdi,%rax), %r10
        imul %r11, %r10
        mov %r10, (%rdi,%rax)
        sub $4, %rax
        jz loop
```

• **Schedule (2 cycles per iteration)**
Loop Example

- 4 iterations are overlapped
  - value of \%r11 don’t change
  - 4 regs for (\%rdi, \%rax)
  - each addr. incremented by 4*4
  - 4 regs to keep value \%r10
  - Same registers can be reused after 4 of these blocks

generate code for 4 blocks, otherwise need to move

<table>
<thead>
<tr>
<th>mov4</th>
<th>mov2</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov1</td>
<td>mov4</td>
</tr>
<tr>
<td>mul3</td>
<td>jz1</td>
</tr>
<tr>
<td>jz</td>
<td>mul3</td>
</tr>
<tr>
<td>mul2</td>
<td></td>
</tr>
<tr>
<td>sub2</td>
<td></td>
</tr>
<tr>
<td>sub1</td>
<td></td>
</tr>
</tbody>
</table>

loop:

  mov (%rdi,%rax), %r10
  imul %r11, %r10
  mov %r10, (%rdi,%rax)
  sub $4, %rax
  jz loop
Software Pipelining

• Optimal use of resources
• Need a lot of registers
  – Values in multiple iterations need to be kept
• Issues in dependencies
  – Executing a store instruction in an iteration before branch instruction is executed for a previous iteration (writing when it should not have)
  – Loads and stores are issued out-of-order (need to figure-out dependencies before doing this)
• Code generation issues
  – Generate pre-amble and post-amble code
  – Multiple blocks so no register copy is needed
Outline

• Scheduling for loops
• Loop unrolling
• Software pipelining
• Interaction with register allocation
• Hardware vs. Compiler
• Induction Variable Recognition
• loop invariant code motion
Register Allocation and Instruction Scheduling

- If register allocation is before instruction scheduling
  - restricts the choices for scheduling
Example

1: mov 4(%rbp), %rax
2: add %rax, %rbx
3: mov 8(%rbp), %rax
4: add %rax, %rcx
Example

1: mov $4(%rbp), %rax
2: add %rax, %rbx
3: mov $8(%rbp), %rax
4: add %rax, %rcx
Example

1: mov 4(%rbp), %rax
2: add %rax, %rbx
3: mov 8(%rbp), %rax
4: add %rax, %rcx

<table>
<thead>
<tr>
<th>ALUop</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM 1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MEM 2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Example

\begin{align*}
1: & \text{ mov } 4(\%rbp), \%rax \\
2: & \text{ add } \%rax, \%rbx \\
3: & \text{ mov } 8(\%rbp), \%rax \\
4: & \text{ add } \%rax, \%rcx
\end{align*}

Anti-dependence
How about a different register?
Example

1: mov 4(%rbp), %rax
2: add %rax, %rbx
3: mov 8(%rbp), %r10
4: add %r10, %rcx

Anti-dependence
How about a different register?
Example

1: mov 4(%rbp), %rax
2: add %rax, %rbx
3: mov 8(%rbp), %r10
4: add %r10, %rcx
Register Allocation and Instruction Scheduling

- If register allocation is before instruction scheduling
  - restricts the choices for scheduling
Register Allocation and Instruction Scheduling

- If register allocation is before instruction scheduling
  - restricts the choices for scheduling

- If instruction scheduling is before register allocation
  - Register allocation may spill registers
  - Will change the carefully done schedule!!!
Outline

• Scheduling for loops
• Loop unrolling
• Software pipelining
• Interaction with register allocation
• **Hardware vs. Compiler**
• Induction Variable Recognition
• loop invariant code motion
Out Of Order Hardware Scheduling

• Out of order execution
  – If an instruction stalls, go beyond that and start executing non-dependent instructions
  – Pros:
    • Tolerates unpredictable latencies
    • More information available at runtime than compile time
  – Cons:
    • Exploit parallelism only within instruction window
Register Renaming

• Register renaming
  – If there is an anti or output dependency of a register that stalls the pipeline, use a different hardware register
  – Pros:
    • Avoids anti and output dependencies
  – Cons:
    • Cannot do more complex transformations to eliminate dependencies