The recent switch lowering improvements

Hans Wennborg
hwennborg@google.com
A Switch

C:

switch (x) {
  case 0:
    // foo
  case 1:
    // bar
  ...
  default:
    // baz
}

LLVM IR:

switch i32 %x, label %baz [
  i32 0, label %foo
  i32 1, label %bar
  ...
]
A Switch

C:

if (x == 0) {
    // foo
} else if (x == 1) {
    // bar
} else {
    // baz
}

LLVM IR:

switch i32 %x, label %baz [ i32 0, label %foo
    i32 1, label %bar
    ...
]
Lowering

LowerSwitch  SelectionDAGBuilder::visitSwitch
Lowering

LowerSwitch → SelectionDAGBuilder::visitSwitch
Step 0: Cluster adjacent cases
Step 0: Cluster adjacent cases

- 1 to B
- 5 to C
- 2 to B
- 3 to B
- 0 to A
- 4 to C

- 0
- 1-3
- 4-5
Lowering strategies

1. Straight comparisons
2. Jump tables
3. Bit tests
4. Binary search tree
1. Straight comparisons

- \( x = 0 \) → A
- \( 1 \leq x \leq 3 \) → B
- \( 4 \leq x \leq 5 \) → C

Default

- Number of clusters \( \leq 3 \)
2. Bit tests

- Number of destinations ≤ 3
- Range fits in machine word

\[2^0 + 2^3 + 2^6 = 73\]

\[2^1 + 2^4 + 2^7 = 146\]

\[2^2 + 2^5 + 2^8 = 292\]
3. Jump table

**table:**

<table>
<thead>
<tr>
<th>x</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>Default</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
</tr>
</tbody>
</table>

1 ≤ x ≤ 5

- Number of clusters ≥ 4
- Table density ≥ 40%
4. Binary search tree

Bit tests

Jump table

Straight comparisons
4. Binary search tree

- If $x \leq 8$:
  - If $bt(x, 73)$: Go to A
  - If $bt(x, 146)$: Go to B
  - If $bt(x, 292)$: Go to C

- If $x \leq 100$:
  - If $101 \leq x \leq 104$: Go to table[x-101]
  - Default

- If $x \leq 999$:
  - If $x = 1000$: Go to H
  - If $x = 2000$: Go to I
  - If $x = 3000$: Go to J

- Default
What changed?
Old algorithm: top-down

- Consider the range of cases
- Lower by cmps, bit tests or jump table? If yes, done
- Split the range in two*, creating BST
- Repeat for both sides
Old algorithm: pivot selection is hard

* Pivot heuristic: maximize gap size and sum density of LHS and RHS.

Heuristic helps find jump tables
But trees might not be balanced
(PR22262)
New algorithm: bottom-up

- Consider the whole range of cases
- Find case clusters suitable for bit tests
- Find case clusters suitable for jump tables
- Build binary search tree
New algorithm: benefits

- Lowering strategies decoupled
  a. Code is easier to follow
  b. Can do less work at -O0
- Jump table extraction is optimal*
- BST will be balanced**

* For our size and density criteria
** Next slide!
Balanced by node count

- $x < 40$
  - $x < 20$
    - $x = 0$
    - $x = 10$
  - $x < 60$
    - $x = 20$
    - $x = 30$
    - $x = 40$
    - $x = 50$
    - $x = 60$
    - $x = 70$
Balanced by node weight

0 10 20 30 40 50 60 70

x < 50

x < 10

x = 0

x < 30

x = 10

x = 20

x = 30

x = 40

x = 50

x = 60

x = 70
Balanced by node weight

```
x < 50
  x < 10
    x = 0
    x = 10
    x = 20
  x < 30
    x = 10
    x = 30
    x = 40
  x = 70
    x = 50
    x = 60
```

<table>
<thead>
<tr>
<th>x</th>
<th>Branches</th>
<th>x weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>70</td>
<td>2</td>
<td>2000</td>
</tr>
</tbody>
</table>

(Sum: 2055) (Without weight balancing: 3052)
Summary

- Trees are balanced
- Jump tables are found
- Uses profile info