Linear Scan Register Allocation

on Static Single Assignment Form

Christian Wimmer
cwimmer@uci.edu
www.christianwimmer.at
April 2010

Department of Computer Science
University of California, Irvine
Introduction

- Register allocation
  - Graph coloring algorithm
  - Linear scan algorithm

- Static single assignment (SSA) form
  - One definition per variable that dominates all uses
  - Variables that interfere somewhere also interfere at one definition
  - Interference graph is chordal
  - Graph coloring in polynomial time

- Linear scan algorithm on SSA form
  - Liveness analysis without iterative data flow analysis
  - Use SSA properties during register allocation
  - SSA deconstruction integrated with resolution phase of linear scan
Java HotSpot™ Client Compiler

Bytecode Parsing

- Bytecodes
  - Method Inlining
  - Constant Propagation
  - Local Value Numbering
- HIR (SSA Form)
  - Null Check Elimination
  - Global Value Numbering
- Optimized HIR

Global Optimizations

Back End

- LIR Generation
  - LIR
  - Register Allocation
  - Code Generation
  - Machine Code
Phases of Linear Scan Algorithm

Linear Scan not on SSA Form

SSA Form Deconstruction

LIR not in SSA Form

Lifetime Intervals

Splitting and Spilling of Intervals

Registers Assigned to Intervals

LIR not in SSA Form

Requires a Data Flow Analysis

LIR Generation

LIR in SSA Form

Lifetime Intervals

No Iterative Data Flow Analysis

Lifetime Intervals

Linear Scan Algorithm

Splitting and Spilling of Intervals

Registers Assigned to Intervals

Resolution

SSA Form Deconstruction

LIR not in SSA Form
Lifetime Intervals Without SSA Form

define R10, R11
20: move 1 -> R12
22: move R11 -> R13
24: label B2
26: cmp R13, 1
28: branch lessThan B4
30: label B3
32: mul R12, R13 -> R14
34: sub R13, 1 -> R15
36: move R14 -> R12
38: move R15 -> R12
40: jump B2
42: label B4
use R10, R12

Lifetime Interval
Definition position
Use position
Lifetime Intervals With SSA Form

20: label B2
phi [1, R14] -> R12
phi [R11, R15] -> R13
22: cmp R13, 1
24: branch lessThan B4
26: label B3
28: mul R12, R13 -> R14
30: sub R13, 1 -> R15
32: jump B2
34: label B4
use R10, R12

---

define R10, R11

Lifetime Intervals With SSA Form

- Lifetime Interval
- Definition position
- Use position
Construction of Lifetime Intervals

Initial Live Set from Successors
Add Input Operands of Successors’ Phis
Process Operations in Reverse Order
Remove Phi Functions from Live Set
Extend Live Ranges of Loop Variables

20: label B2
   phi [1, R14] -> R12
   phi [R11, R15] -> R13
22: cmp R13, 1
24: branch lessThan B4
26: label B3
28: mul R12, R13 -> R14
30: sub R13, 1 -> R15
32: jump B2
34: label B4
   use R10, R12

define R10, R11
Irreducible Control Flow

phi \([R10, R12] \rightarrow R11\)

phi \([R10, R11] \rightarrow R12\)

phi \([R20, R22] \rightarrow R21\)

phi \([R20, R21] \rightarrow R22\)
Changes to Linear Scan Algorithm

Linear scan not on SSA form

- Interval $i10$ has a lifetime hole.
- intervals $i10$ and $i11$ can intersect.

Without SSA form: Intervals that are currently not live can block registers.

Linear scan on SSA form

- Interval $i10$ has a lifetime hole.
- intervals $i10$ and $i11$ never intersect.

SSA form guarantees: Intervals that are currently not live never block registers.
SSA Deconstruction during Resolution

**Resolution**
Visit intervals live across control-flow edges

**SSA Deconstruction**
Also visit intervals starting at the control-flow edge

![Diagram showing SSA Deconstruction](image)

**B2 – B4:**
No move necessary

**B3 – B4:**
move s1 -> eax
move s2 -> ecx

phi [R13, R12] -> R14
Compilation Time

Compilation time of baseline and SSA form version of linear scan

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>SSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECjvm2008</td>
<td>23%</td>
<td>18%</td>
</tr>
<tr>
<td>SPECjbb2005</td>
<td>23%</td>
<td>18%</td>
</tr>
<tr>
<td>DaCapo</td>
<td>30%</td>
<td>28%</td>
</tr>
<tr>
<td>SciMark</td>
<td>24%</td>
<td>17%</td>
</tr>
</tbody>
</table>

2 * Intel Xeon X5140, 2.33 GHz, 4 cores, 32 GByte memory
Ubuntu Linux, kernel version 2.6.28
SPECjvm2008: Lagom w/o SciMark
## Phi Functions and Move Instructions

<table>
<thead>
<tr>
<th></th>
<th>DaCapo</th>
<th>SciMark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>SSA Form</td>
</tr>
<tr>
<td><strong>Before Register Allocation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moves</td>
<td>402,678</td>
<td>355,936</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phi Functions</td>
<td>0</td>
<td>20,542</td>
</tr>
<tr>
<td><strong>After Register Allocation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moves Register to Register</td>
<td>127,318</td>
<td>124,351</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moves Constant to Register</td>
<td>71,967</td>
<td>70,663</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moves Stack to Register</td>
<td>3,718</td>
<td>3,722</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moves Register to Stack</td>
<td>65,973</td>
<td>56,639</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moves Constant to Stack</td>
<td>0</td>
<td>1,386</td>
</tr>
<tr>
<td>Moves Stack to Stack</td>
<td>0</td>
<td>647</td>
</tr>
</tbody>
</table>
Summary

- Linear scan algorithm on SSA form
  - Liveness analysis without iterative data flow analysis
  - Use SSA properties during register allocation
  - SSA deconstruction integrated with resolution phase of linear scan

- Benefits
  - Faster, especially liveness analysis
  - Simpler compiler code
  - Equally good (or slightly better) machine code
  - Eliminates SSA deconstruction phase

- Do register allocation on SSA form!
  - No matter what algorithm you use