Linear Scan Register Allocation on SSA Form

Christian Wimmer  Michael Franz
UC, Irvine
Register Allocation & SSA

• Register allocation
  – Definition: assigning physical registers to local variables and temporary values (virtual registers)
  – NP-complete
  – Simple and fast algorithms: linear scan
  – Slow algorithms: graph coloring, puzzle solving

• SSA form
  – A type of intermediate representation
  – All variables have only a single point of definition
  – Phi functions are inserted to merge different variables of the predecessor blocks
Linear Scan Register Allocation

- **Traditional linear scan register allocation**
  - Low compilation overhead
  - Generated code comparable with graph coloring
  - Performs SSA form deconstruction before RA
  - Interval intersection has to be performed through data flow analysis
  - Widely used in VM’s such as Java HotSpot, Jikes RVM, LLVM 2.9

- **Linear scan on SSA form**
  - Need no SSA form destruction before RA
  - Lifetime interval construction is simplified
  - Faster than traditional linear scan
  - Generated code is equally good or slightly better
Linear Scan Register Allocation not on SSA

Figure 1. Linear scan register allocation not on SSA form.
Linear Scan Register Allocation on SSA

LIR Construction

LIR in SSA Form with Virtual Registers

Lifetime Analysis

No Data Flow Analysis

Lifetime Intervals with Lifetime Holes

Linear Scan Algorithm

Splitting and Spilling of Lifetime Intervals

Registers Assigned to Lifetime Intervals

Includes SSA Form Deconstruction

Resolution

LIR not in SSA Form with Physical Registers

Figure 2. Linear scan register allocation on SSA form.
Lifetime Intervals and SSA Form

• The first step is to construct lifetime intervals for variables

• Lifetime intervals on SSA form provides two advantages:
  – No artificial order is imposed for moves resulting from phi functions, resulting in more freedom for the register allocator;
  – The lifetime intervals for phi functions have fewer lifetime holes, leading to less state changes of the intervals during allocation.
Lifetime Intervals and SSA Form

\textit{define \textit{R10} and \textit{R11}}

\texttt{\textbf{20:} move 1 \rightarrow R12}
\texttt{\textbf{22:} move R11 \rightarrow R13}

\texttt{\textbf{24:} label B2}
\texttt{\textbf{26:} cmp R13, 1}
\texttt{\textbf{28:} branch lessThan B4}

\texttt{\textbf{30:} label B3}
\texttt{\textbf{32:} mul R12, R13 \rightarrow R14}
\texttt{\textbf{34:} sub R13, 1 \rightarrow R15}
\texttt{\textbf{36:} move R14 \rightarrow R12}
\texttt{\textbf{38:} move R15 \rightarrow R13}
\texttt{\textbf{40:} jump B2}
\texttt{\textbf{42:} label B4}
\texttt{\texttt{\textit{use \textit{R10} and \textit{R12}}}}

\textit{define \textit{R10} and \textit{R11}}

\texttt{\textbf{20:} label B2}
\texttt{\textbf{22:} cmp R13, 1}
\texttt{\textbf{24:} branch lessThan B4}

\texttt{\texttt{\textit{phi [1, R14] \rightarrow R12}}}
\texttt{\texttt{\textit{phi [R11, R15] \rightarrow R13}}}

\texttt{\textbf{26:} label B3}
\texttt{\textbf{28:} mul R12, R13 \rightarrow R14}
\texttt{\textbf{30:} sub R13, 1 \rightarrow R15}
\texttt{\textbf{32:} jump B2}
\texttt{\texttt{\textit{use \textit{R10} and \textit{R12}}}}

(a) LIR without SSA form

(b) LIR with SSA form

(c) Lifetime intervals without SSA form
Lifetime Intervals and SSA Form

**define R10 and 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 -> R13  
40: jump B2  

42: label B4  
use R10 and R12

**define R10 and R11**

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 and R12

(a) LIR without SSA form  
(b) LIR with SSA form

(d) Lifetime intervals with SSA form
Lifetime Analysis

• Linear scan algorithm does not operate on a structured control flow graph, but on a linear list of blocks

• The block order has a high impact on the quality and speed of linear scan: a good block order leads to short lifetime intervals with few holes

• Block order properties for SSA form LSRA:
  – All predecessors of a block are located before this block with the exception of backward edges of loops
  – All blocks that are part of the same loop are contiguous
Lifetime Analysis

• Input:
  – Intermediate representation in SSA form. An operation has input and output operands.
  – A linear block order where all dominators of a block are before this block, and where all blocks belonging to the same loop are contiguous.

• Output:
  – One lifetime interval for each virtual register, covering operation numbers where this register is alive and with lifetime holes in between
Algorithm for constructing lifetime intervals

```
BUILDINTERVALS
for each block b in reverse order do
    live = union of successor.b live
    for each successor of b
        for each phi function phi of successors of b do
            live.add(phi.inputOf(b))
        for each opd in live do
            intervals[opd].addRange(b.from, b.to)
        for each operation op of b in reverse order do
            for each output operand opd of op do
                intervals[opd].setFrom(op.id)
                live.remove(opd)
            for each input operand opd of op do
                intervals[opd].addRange(b.from, op.id)
                live.add(opd)
        for each phi function phi of b do
            live.remove(phi.output)

if b is loop header then
    loopEnd = last block of the loop starting at b
    for each opd in live do
        intervals[opd].addRange(b.from, loopEnd.to)

b.liveln = live
```

Figure 4. Algorithm for construction of lifetime intervals.
 define R10 and 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 -> R13
40: jump B2
42: label B4
 use R10 and R12

 define R10 and R11
20: label B2
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 and R12

(a) LIR without SSA form
(b) LIR with SSA form

d) Lifetime intervals with SSA form
Linear Scan Algorithm

• The main linear scan algorithm needs no modification to work on SSA form
• It processes the lifetime intervals sorted by their start position and assigns a register or stack slot to each interval
• Four sets of intervals are managed:
  – Unhandled, contains the intervals that start after the current position
  – Active, contains the intervals that are live at the current position
  – Inactive, contains the intervals that start before and end after the current position, but that have a lifetime hole at the current position
  – Handled, contains the intervals that end before the current position
Linear Scan Algorithm

TRYALLOCATEFREEREG
set freeUntilPos of all physical registers to maxInt
for each interval it in active do
  freeUntilPos[it.reg] = 0
for each interval it in inactive intersecting with current do
  freeUntilPos[it.reg] = next intersection of it with current
reg = register with highest freeUntilPos
...

ALLOCATEBLOCKEDREG
set nextUsePos of all physical registers to maxInt
for each interval it in active do
  nextUsePos[it.reg] = next use of it after start of current
for each interval it in inactive intersecting with current do
  nextUsePos[it.reg] = next use of it after start of current
reg = register with highest nextUsePos
...

Figure 6. Algorithm for register selection (from [30]).
Evaluation

• The client compiler of Sun’s Java HotSpot VM has been modified

• The product version is used as the baseline
  – Highly tuned product version

• Four benchmarks:
  – SPECjvm2008
  – SPECjbb2005
  – DaCapo
  – SciMark
### Comparison of statistics

<table>
<thead>
<tr>
<th></th>
<th>SPECjvm2008</th>
<th></th>
<th>SPECjbb2005</th>
<th></th>
<th>DaCapo</th>
<th></th>
<th>SciMark</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>SSA Form</td>
<td>Baseline</td>
<td>SSA Form</td>
<td>Baseline</td>
<td>SSA Form</td>
<td>Baseline</td>
<td>SSA Form</td>
</tr>
<tr>
<td>Compilation Statistics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiled Methods</td>
<td>6,788</td>
<td>6,813</td>
<td>521</td>
<td>520</td>
<td>8,242</td>
<td>8,242</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>Compiled Bytecodes [KByte]</td>
<td>1,094</td>
<td>1,098</td>
<td>78</td>
<td>78</td>
<td>2,272</td>
<td>2,275</td>
<td>3.64</td>
<td>3.65</td>
</tr>
<tr>
<td>Compilation Time [msec.]</td>
<td>4,250</td>
<td>4,080</td>
<td>-4%</td>
<td>287</td>
<td>275</td>
<td>-4%</td>
<td>13,390</td>
<td>12,700</td>
</tr>
<tr>
<td>Back End Time [msec.]</td>
<td>1,170</td>
<td>1,020</td>
<td>-13%</td>
<td>82</td>
<td>71</td>
<td>-13%</td>
<td>2,930</td>
<td>2,460</td>
</tr>
<tr>
<td>Machine Code Size [KByte]</td>
<td>4,581</td>
<td>4,563</td>
<td>-0%</td>
<td>404</td>
<td>401</td>
<td>-1%</td>
<td>11,760</td>
<td>11,719</td>
</tr>
<tr>
<td>Memory Allocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lifetime Analysis [KByte]</td>
<td>65,248</td>
<td>58,877</td>
<td>-10%</td>
<td>5,047</td>
<td>4,559</td>
<td>-10%</td>
<td>171,650</td>
<td>129,794</td>
</tr>
<tr>
<td>Allocation and Resolution [KByte]</td>
<td>48,171</td>
<td>48,169</td>
<td>-0%</td>
<td>3,255</td>
<td>3,239</td>
<td>-0%</td>
<td>89,144</td>
<td>88,879</td>
</tr>
<tr>
<td>LIR Before Register Allocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moves</td>
<td>203,671</td>
<td>180,640</td>
<td>-11%</td>
<td>15,797</td>
<td>13,644</td>
<td>-14%</td>
<td>402,678</td>
<td>356,936</td>
</tr>
<tr>
<td>Phi Functions</td>
<td>0</td>
<td>10,689</td>
<td></td>
<td>0</td>
<td>973</td>
<td></td>
<td>0</td>
<td>20,542</td>
</tr>
<tr>
<td>LIR After Register Allocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moves Register to Register</td>
<td>55,592</td>
<td>53,856</td>
<td>-3%</td>
<td>4,473</td>
<td>4,245</td>
<td>-5%</td>
<td>127,318</td>
<td>124,351</td>
</tr>
<tr>
<td>Moves Constant to Register</td>
<td>35,348</td>
<td>34,612</td>
<td>-2%</td>
<td>3,129</td>
<td>3,028</td>
<td>-3%</td>
<td>71,967</td>
<td>70,663</td>
</tr>
<tr>
<td>Moves Stack to Register</td>
<td>4,537</td>
<td>4,550</td>
<td>+0%</td>
<td>335</td>
<td>335</td>
<td>0%</td>
<td>3,718</td>
<td>3,722</td>
</tr>
<tr>
<td>Moves Register to Stack</td>
<td>38,715</td>
<td>33,650</td>
<td>-13%</td>
<td>2,636</td>
<td>2,187</td>
<td>-17%</td>
<td>65,973</td>
<td>56,639</td>
</tr>
<tr>
<td>Moves Constant to Stack</td>
<td>0</td>
<td>926</td>
<td></td>
<td>0</td>
<td>105</td>
<td></td>
<td>0</td>
<td>1,386</td>
</tr>
<tr>
<td>Moves Stack to Stack</td>
<td>0</td>
<td>294</td>
<td></td>
<td>0</td>
<td>22</td>
<td></td>
<td>0</td>
<td>647</td>
</tr>
</tbody>
</table>

*Figure 9. Comparison of compilation statistics.*
Impact on Compile Time

Figure 10. Compilation time of baseline (B) and SSA form (S) version of linear scan.
Impact on Run time

• The impact on run time is low
  – No significant difference

• Reasons
  – The main allocation algorithm is unchanged
  – The same spilling decisions are made with and without SSA form
  – The speedups are generally below the random noise
  – Only the FFT benchmark of SciMark has a speedup of 1%
Critique

- Very well-written paper
- I enjoyed reading it
- The details are presented clearly
- Experimental results are thorough and convincing
- The idea is not really innovative
  - A mature algorithm implemented on SSA form instead of non-SSA form