Register Allocation for Compressed ISAs in LLVM

Andreas Fried, Maximilian Stemmer-Grabow, Julian Wachter | 25 February 2023
Compressed RISC Instruction Sets
RISC-V extension C (RVC)

Uncompressed arithmetic instruction

Compressed arithmetic instruction

- Shorter encodings (16 bit) for most important/frequent instructions
- Only 8 registers (s0 – s1, a0 – a5), only two-address-form
- Can be mixed with uncompressed instructions

- Improved *code density* (“functionality per byte”)
- Important for embedded applications

Also offered by ARM32 (Thumb), MIPS (microMIPS), PowerPC (VLE extension), ARCompact, . . .
Compilation Flow

Assembler handles compression transparently

Compiler: Ensure many instructions fulfill requirements ⇒ Register allocator should be aware
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA
  (defs on different paths)
- no re-assignments
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA (defs on different paths)
- no re-assignments

Priority
- Inter-block by size (instructions spanned)
- Intra-block top → bottom
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA
  (defs on different paths)
- no re-assignments

Priority
- Inter-block by size
  (instructions spanned)
- Intra-block top → bottom

Assume 3 registers: ▢ ▢ ▢
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA (defs on different paths)
- no re-assignments

Priority
- Inter-block by size (instructions spanned)
- Intra-block top→bottom

Assume 3 registers: 

4/10  2021-02-25  Andreas Fried: Register Allocation for Compressed ISAs in LLVM
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA
  (defs on different paths)
- no re-assignments

Priority
- Inter-block by size
  (instructions spanned)
- Intra-block top → bottom

Assume 3 registers: 🔴 🔵 🔳
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA (defs on different paths)
- no re-assignments

Priority
- Inter-block by size (instructions spanned)
- Intra-block top → bottom

Assume 3 registers: ▢ □ ▪
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA (defs on different paths)
- no re-assignments

Priority
- Inter-block by size (instructions spanned)
- Intra-block top→bottom

Assume 3 registers: ⬤ ⬤ ⬤
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA
  (defs on different paths)
- no re-assignments

Priority
- Inter-block by size
  (instructions spanned)
- Intra-block top → bottom

Assume 3 registers: 🔴 🔵 🔴
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA (defs on different paths)
- no re-assignments

Priority
- Inter-block by size (instructions spanned)
- Intra-block top → bottom

Assume 3 registers:
LLVM “Greedy” Register Allocator

Live-intervals
- Symbolic values
- not quite SSA (defs on different paths)
- no re-assignments

Priority
- Inter-block by size (instructions spanned)
- Intra-block top → bottom

Assume 3 registers: 

4/10 2021-02-25 Andreas Fried: Register Allocation for Compressed ISAs in LLVM
Cost-Per-Use

More than one available register, which to choose?

All registers are equal but . . .

- Callee-saves need to be spilled ⇒ Extra cost on first use
- Some registers are to be avoided ⇒ Extra cost on every use
Cost-Per-Use

More than one available register, which to choose?

All registers are equal but . . .

- Callee-saves need to be spilled ⇒ Extra cost on first use
- Some registers are to be avoided ⇒ Extra cost on every use (e.g. non-compressible)
Cost-Per-Use

More than one available register, which to choose?

All registers are equal but . . .
- Callee-saves need to be spilled ⇒ Extra cost on first use
- Some registers are to be avoided ⇒ Extra cost on every use (e.g. non-compressible)

Already a mechanism to prefer compressible registers but . . .
- is there even a compressible encoding? (mul, div, FP arithmetic)
- two-address-form? (sub a0, a1, a5)
- compressible registers for other arguments? (sub a0, a0, a6)

Need to be aware of specific circumstances
When (Not) To Compress

Look at *potentially compressible* instructions

- or a0, ?, ?  ld a3, ?(fp)  st ?, ?(?)  ✓
- and ?, ?, a6  sub a0, a1, ?  mul ?, ?, ?  ✗

Define live-interval's *compressibility* $C(L)$: number of potentially compressible instructions using or defining $L$

- context-sensitive measure where to use compressible registers
- better approximation as other live-intervals are assigned
When (Not) To Compress

Look at *potentially compressible* instructions

\[
\text{or } a0, ?, ? \quad \text{ld } a3, ?(fp) \quad \text{st } ?, ?(\?) \quad \checkmark \\
\text{and } ?, ?, a6 \quad \text{sub } a0, a1, ? \quad \text{mul } ?, ?, ? \quad \times
\]

Define live-interval’s *compressibility* $C(L)$: number of potentially compressible instructions using or defining $L$

- **context-sensitive** measure where to use compressible registers
- better approximation as other live-intervals are assigned

\[
\begin{align*}
\text{double } a &= q->a; \quad \text{fld } %a, a_{-}\text{offset}(\%q) \quad C(\%q) = 3 \\
\text{double } b &= q->b; \quad \text{fld } %b, b_{-}\text{offset}(\%q) \quad C(\%a) = 1 \\
q->c &= (a + b) \ast (a - b); \quad \text{fadd.d } %t1, %a, %b \quad C(\%b) = 1 \\
&\quad \text{fsub.d } %t2, %a, %b \quad C(\%c) = 1 \\
&\quad \text{fmul.d } %c, %t1, %t2 \quad C(\%t1) = 0 \\
&\quad \text{fsd } %c, c_{-}\text{offset}(\%q) \quad C(\%t2) = 0
\end{align*}
\]

Focus on these
Register Allocator Adaptations
Priority adjustment
Boost priority of live-ranges with high compressibility
Register Allocator Adaptations

1. **Priority adjustment**
   Boost priority of live-ranges with high compressibility

2. **Register selection**
   Choose register with highest potential compressibility

1. Priority adjustment
   Boost priority of live-ranges with high compressibility

2. Register selection
   Choose register with highest potential compressibility
Register Allocator Adaptations

1. **Priority adjustment**
   Boost priority of live-ranges with high compressibility

2. **Register selection**
   Choose register with highest potential compressibility

3. **Choice of Evictee**
   Consider difference in compressibility
Evaluation

Benchmarks
- C/C++ benchmarks from SPEC CPU2000 and CPU2006

Binary size
- Compare size of unlinked object files with old/new register allocation
- Size reduction 1.93% in the mean, up to 6.5%

Performance impact
- CPU2000 on “VisionFive” SBC (SiFive U74 core)
- runtime changes up to −1.1%/ + 1.5%, geo. mean +0.3%
- weak/no correlation with spill costs ⇒ probably random fluctuations
Compression Results

- integer
- floating-point
- geo. mean 1.93%
Conclusion

- Compression-aware register allocation reduces binary size while being performance-neutral
- Especially where not many types of instructions are compressible
- Future: Opportunity for less regular, more specialized compression schemes

Register Allocator Adaptations

1. Priority adjustment
   - Boost priority of live-ranges with high compressibility
2. Register selection
   - Choose register with highest potential compressibility
3. Choice of Evictee
   - Consider difference in compressibility

Compression Results
END
Phases
Priority Adjustment

- Boost intervals with higher compressibility $C_{rel}(L)$ relative to their size $S(L)$

$$C_{rel}(L) = \frac{\text{# potentially compressible instructions in } L}{\text{# uses and defs in } L} \quad 0 \leq C_{rel}(L) \leq 1$$

- fully compressible $\Rightarrow$ full priority
  
  not compressible at all $\Rightarrow$ reduce priority by factor $\alpha$

$$P(L) = S(L) \cdot ((1 - \alpha) + \alpha C_{rel}(L))$$

$$P(L) = (1 - \alpha) S(L)$$
## Compression-Aware Register Selection

- Status quo: Choose first register that is
  - free
  - has no cost-per-use
- New: How much potential compression if $L \mapsto r$ chosen?
  - Still avoid first use of callee-saves, no other cost-per-use

<table>
<thead>
<tr>
<th></th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
<th>r6</th>
<th>r7</th>
<th>r8</th>
</tr>
</thead>
<tbody>
<tr>
<td>free?</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>cost-per-use?</td>
<td>—</td>
<td>✓</td>
<td>X</td>
<td>—</td>
<td>X</td>
<td>✓</td>
<td>—</td>
<td>X</td>
</tr>
<tr>
<td>$C(L \mapsto r)$</td>
<td>—</td>
<td>4</td>
<td>7</td>
<td>—</td>
<td>18</td>
<td>20</td>
<td>—</td>
<td>10</td>
</tr>
</tbody>
</table>

↑ old choice  ↑ new choice
Choice Of Evictees

Assign \( L \mapsto r \) and evict all interfering \( L'_1, L'_2, \ldots \in E(L, r) \)?

If so, which \( r \)?

- Look at spill weight (cost to spill and reload) \( w(L) \) vs. \( w(L'_i) \)
- If any \( w(L'_i) > w(L) \), don’t evict
- Status quo: Choose \( r \) to minimize \( \max w(L'_i) \)
- New: Also consider difference in compression \( \sum_{\text{lost}} C(L'_i) - C(L \mapsto r) \)

\[
\text{cost}(L, r) = (1 - \beta) \text{maxweight} + \beta \Delta_{\text{compression}}
\]
Groups
Compression Advantages & Further Opportunities

Floating Point
- only load/store compressible
- compiler did not know
- better heuristic was required

Integer
- existing heuristic OK (good structure of RVC)
- improvements possible
- missing compression mostly due to three-address-form

Constants, Jumps & Branches
- immediate range more important
- life-intervals not worthwhile ⇒ de-prioritized

![Diagram showing the share of compressed instructions for different operation groups.]
Performance
Performance Impact

Expectation: Not much change
- Cache model: < 1% improvement
- “Inconsequential” changes: ± 2% variance

Measurement
- CPU2000 on “VisionFive” SBC (SiFive U74)
- 10 runs, relative standard deviation $\sigma_r < 1.8\%$
- Welch’s t-test at $p = 0.05$, inconclusive results marked “∗”

Results
- 4 benchmarks faster, 4 slower, 6 inconclusive
- Mean over all: 0.3% slowdown
Possible systemic changes
- more/worse spilling with new eviction?
- more copies?

Measuring
- use LLVM spill cost analysis
- correlate performance with spill cost

Result: correlation is weak
- spills: $r^2 = 12.2\%$
- reloads: $r^2 = 9.3\%$
- copies: $r^2 = 6.0\%$

$\Rightarrow$ Performance changes are probably random