Karlsruhe Institute of Technology

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

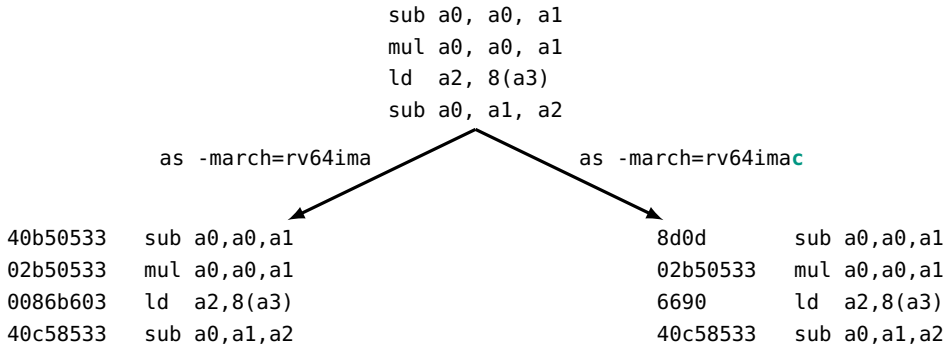
Compressed arithmetic instruction

- 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

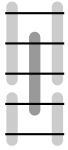


- Assembler handles compression transparently
- Compiler: Ensure many instructions fulfill requirements ⇒ **Register allocator should be aware**

LLVM “Greedy” Register Allocator

Live-intervals

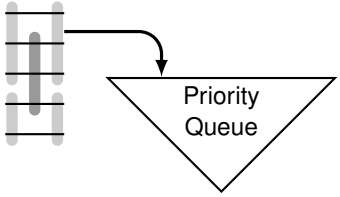
- Symbolic values
- not quite SSA
  (defs on different paths)
- no re-assignments



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

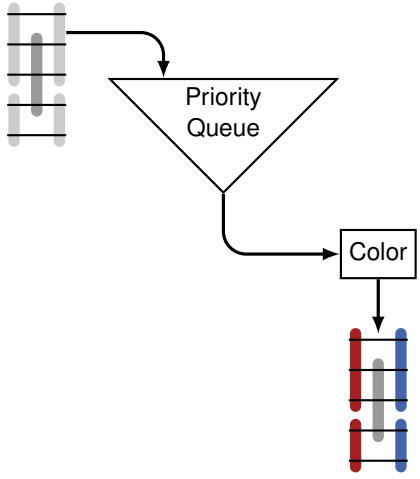
Live-intervals

- 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

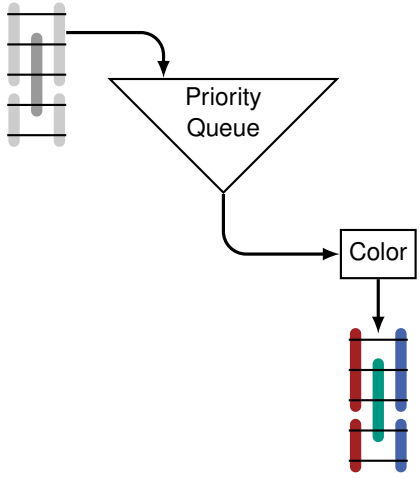
Live-intervals

- 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:



LLVM “Greedy” Register Allocator

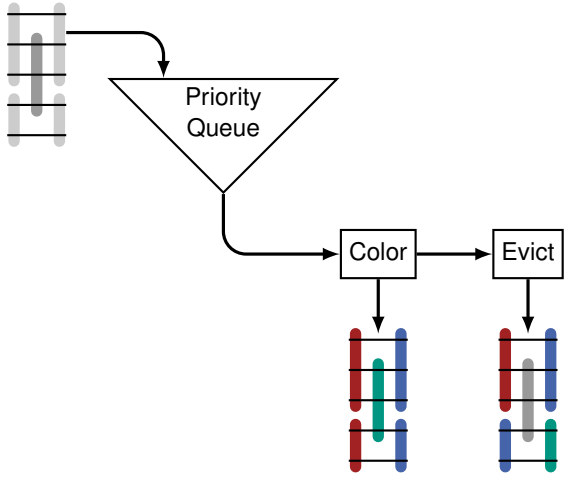
Live-intervals

- 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

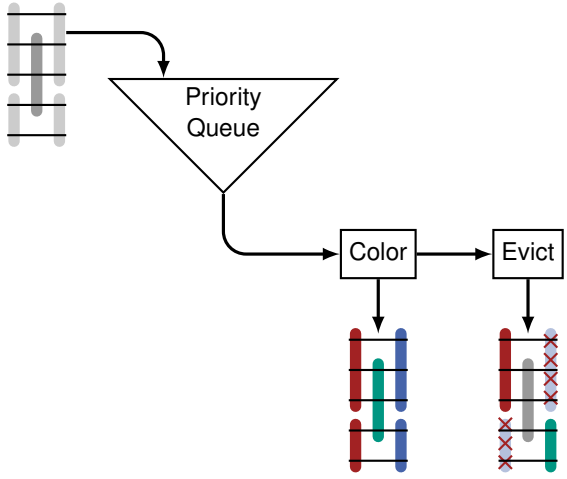
Live-intervals

- 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

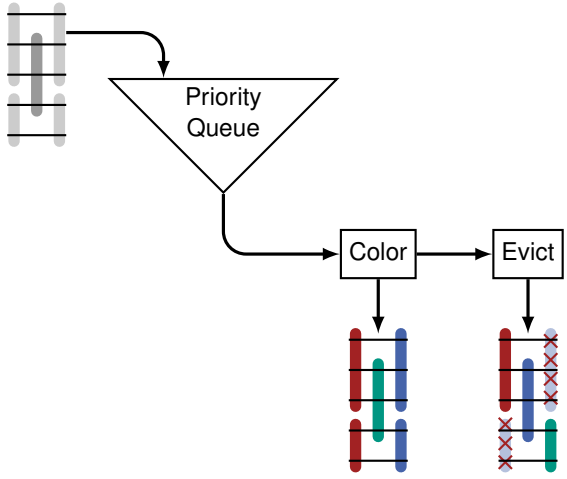
Live-intervals

- 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

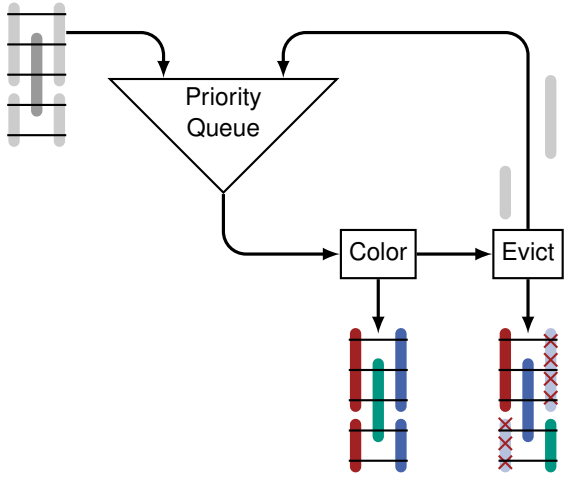
Live-intervals

- 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

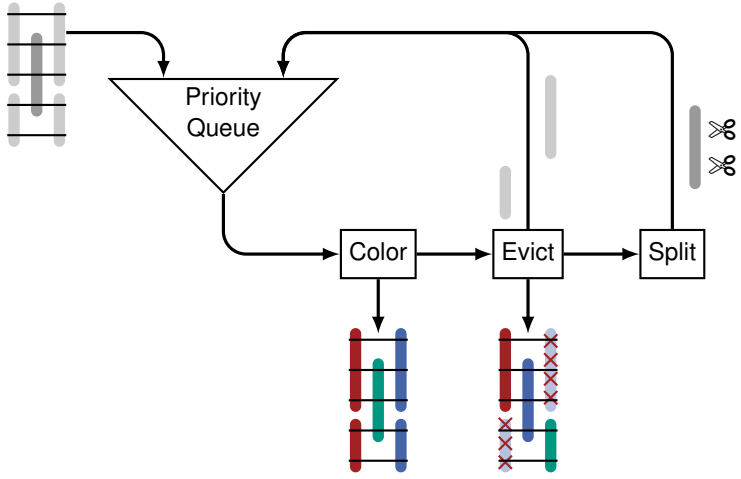
Live-intervals

- 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

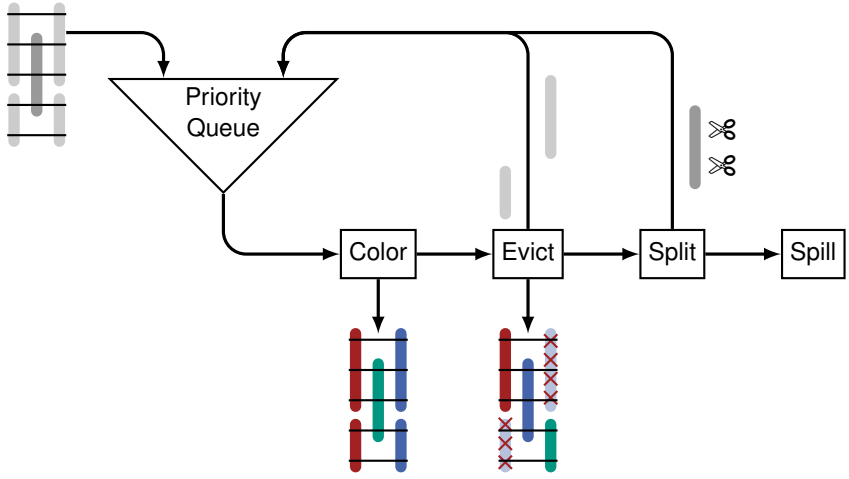
Live-intervals

- 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:



Cost-Per-Use

More than one available register, which to choose?

All registers are equal but . . .

- Callee-saves need to be spilled $\Rightarrow$ Extra cost on *first* use
- Some registers are to be avoided $\Rightarrow$ 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 $\Rightarrow$ Extra cost on *first* use
- Some registers are to be avoided $\Rightarrow$ 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

```
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

```
double a = q->a;
double b = q->b;
q->c = (a + b) * (a - b);
```

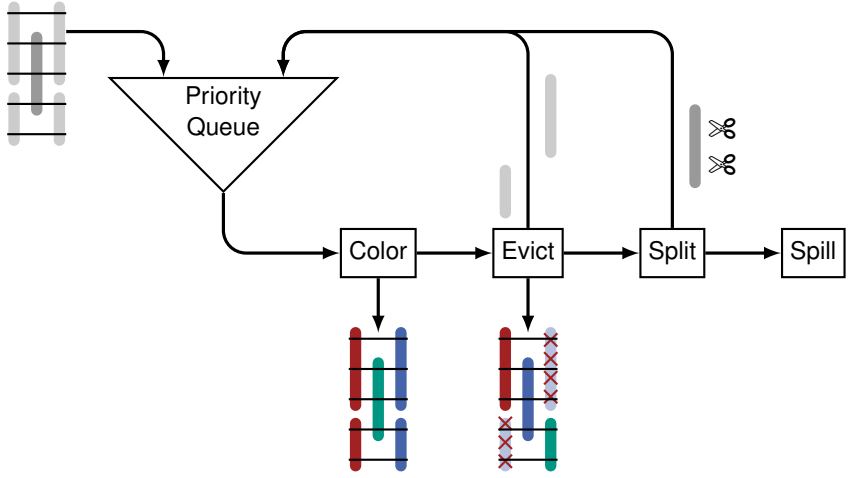
```
fld %a, a_offset(%q)
fld %b, b_offset(%q)
fadd.d %t1, %a, %b
fsub.d %t2, %a, %b
fmul.d %c, % t1, %t2
fsd %c, c_offset(%q)
```

$$\left.\begin{aligned} C(\%q) &= 3 \\ C(\%a) &= 1 \\ C(\%b) &= 1 \\ C(\%c) &= 1 \end{aligned}\right\} \text{Focus on these}$$

$$C(\%t1) = 0$$

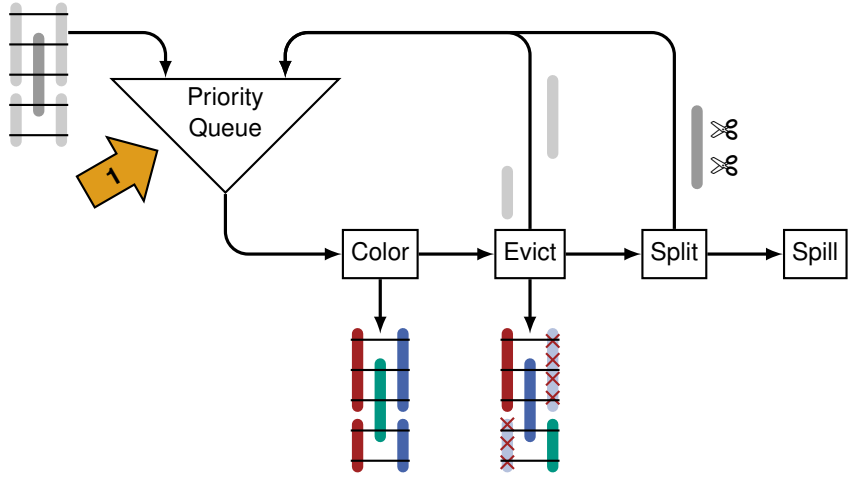
$$C(\%t2) = 0$$

Register Allocator Adaptations



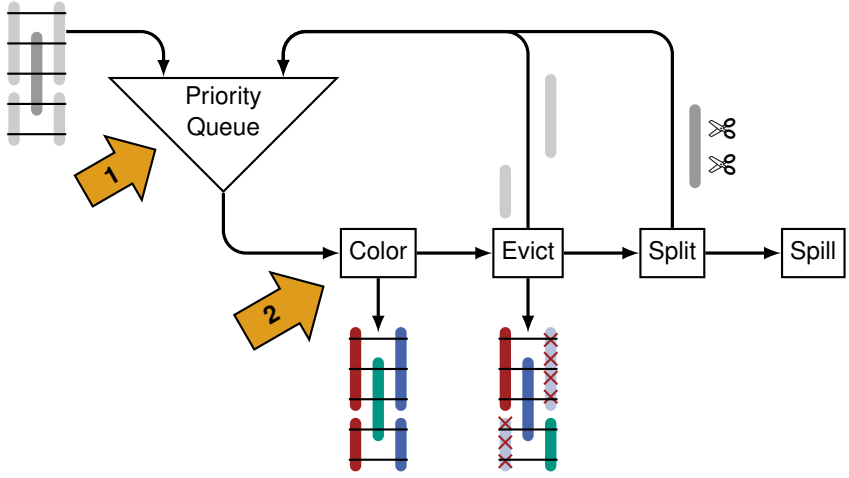
Register Allocator Adaptations

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



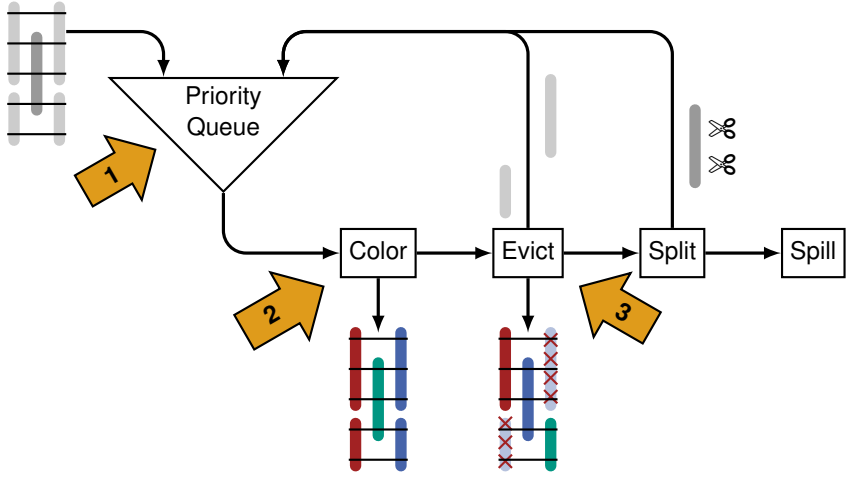
Register Allocator Adaptations

1. **Priority adjustment**
   Boost priority of live-ranges with high compressibility
2. **Register selection**
   Choose register with highest potential compressibility



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

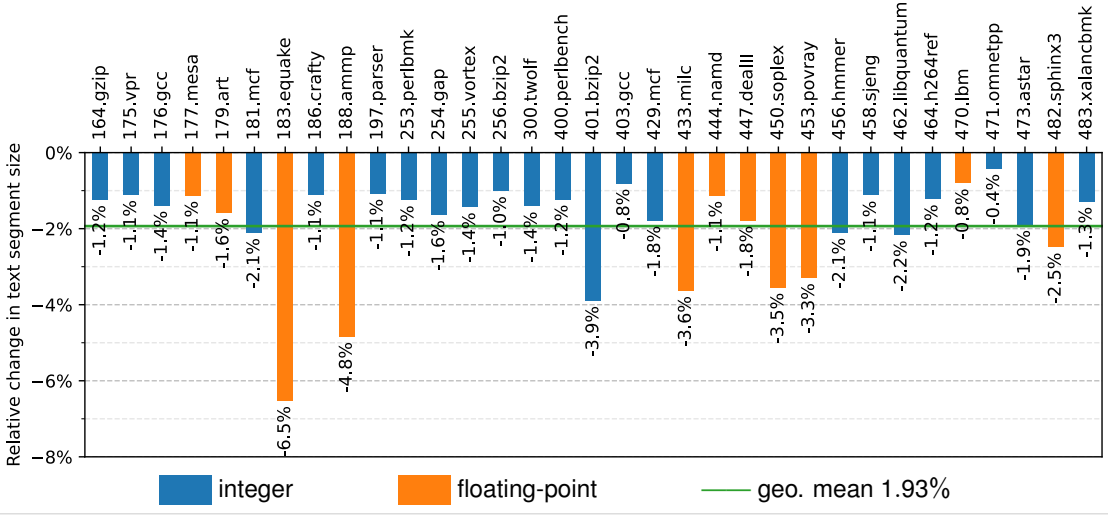
Binary size

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

Performance impact

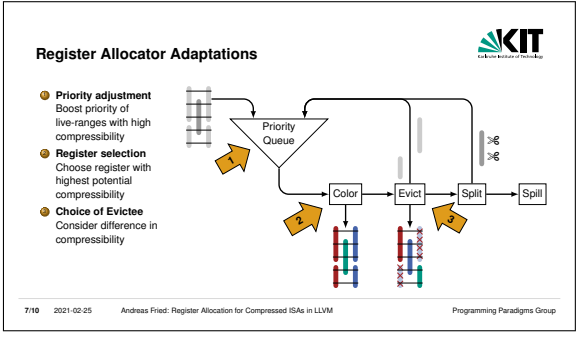
- CPU2000 on “VisionFive” SBC (SiFive U74 core)
- runtime changes up to $-1.1\%/+1.5\%$, geo. mean $+0.3\%$
- weak/no correlation with spill costs $\Rightarrow$ probably random fluctuations

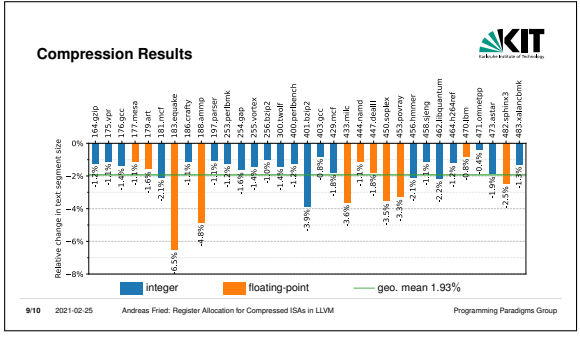
Compression Results



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





END

Phases →

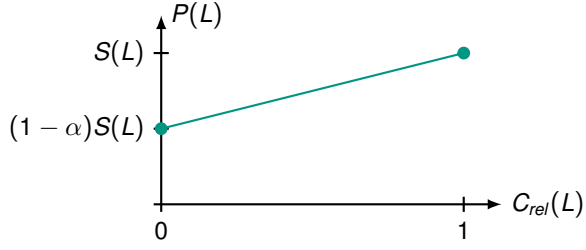
1 Priority Adjustment

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

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

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

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



2 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

	r1	r2	r3	r4	r5	r6	r7	r8
free?	✗	✓	✓	✗	✓	✓	✗	✓
cost-per-use?	—	✓	✗	—	✗	✓	—	✗
$C(L \mapsto r)$	—	4	7	—	18	20	—	10

old choice | new choice

3 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 $\underbrace{\sum C(L'_i)}_{\textit{lost}} - \underbrace{C(L \mapsto r)}_{\textit{gained}}$

$$\textit{cost}(L, r) = (1 - \beta)\textit{maxweight} + \beta\Delta_{\textit{compression}}$$

Groups →

Compression Advantages & Further Opportunities

Floating Point

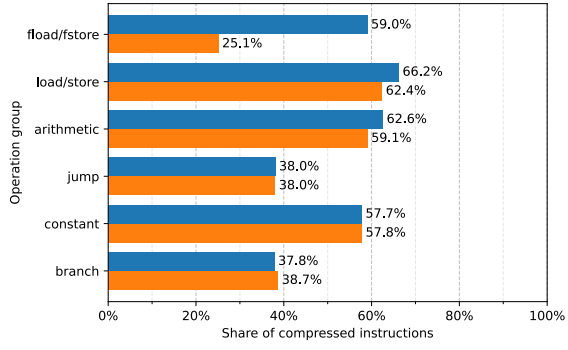
- only load/store compressible
- compiler did not know
- better heuristic was required

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 $\Rightarrow$ de-prioritized



Performance →

Performance Impact

Expectation: Not much change

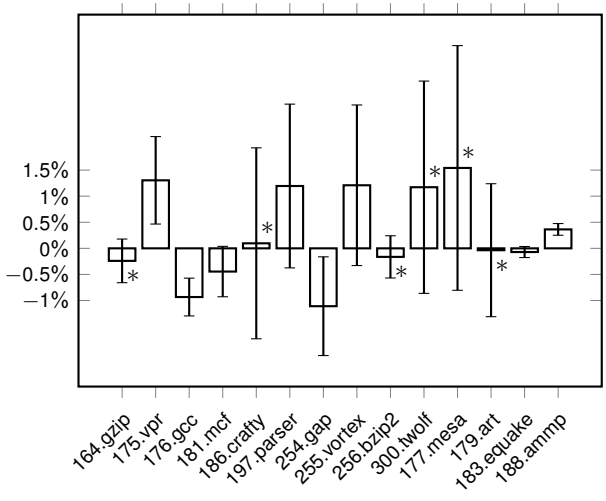
- Cache model: $< 1\%$ improvement
- "Inconsequential" changes: $\pm 2\%$ variance

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



Performance Analysis

Systemic Effect Or Heuristic Fluctuation?

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: $r^2 = 6.0\%$

⇒ **Performance changes are probably random**