Towards a continuous improvement of code-generation for RISC-V...

Philipp Tomsich, VRULL
Life is complicated

Most (published) data for RISC-V is focused on small benchmarks
- Dhrystone
- EEMBC Coremark

This is not surprising, as these benchmarks are well-understood, require for resources and are easy to work with.

As RISC-V starts to target the desktop and servers, we need to expand to cover
- larger benchmarks
- prioritize improvements “where it matters most”

Our focus is SPEC CPU 2017.
SPEC CPU 2017

A large, standardised benchmark suite
- Integer and Floating point
- Single core vs. whole system
- Has built-in validation and comes with well-defined run rules
- Industry standard for “real-world” benchmarking of Linux servers

However
- It comes with a license agreement
- Requires large memory and has considerable runtime
  - ... but this is an area where we can do something about
The competitive landscape

Others (e.g. ARM) had focused efforts to optimize for these benchmarks

- Kyrylo’s talk at the 2019 Cauldron
- ARM’s announcement of auto-vectorization improvements for x264
- Intel’s ICC and AMD’s AOCC have considerable optimisations for this

Life is even harder, as we currently don’t have RVV (which will benefit some of the benchmark components) and Zb[abcs] ratified.
Our methodology

Built on open-source
- QEMU
  - plug-ins to capture dynamic execution profile
  - out-of-band analysis of the captured data
- GCC and LLVM

Analysis happens mainly by hand
- Improvements planned to automate common tasks...

Why QEMU (and not perf)?
- Lack of hardware (especially for non-ratified extensions...)
- Unbeatable performance and access to large main memory
  - We easily run the ‘ref’ workload in for SPEC...
- Unbiased by any specific micro-architecture & allows sharing of data...
Example use-cases

Some of the questions we have started to look into...

- Instruction histograms for the Zb[abcs] instructions
- The expected benefit of CBO.ZERO (and confirming that it works...)
- Code-generation quality in the backend
How Zb[abcs] are we?
Postfix zero-extension... or not.
Is CBO.ZERO beneficial?

`memset` factors prominently on gcc_r

- ~2.7% of dynamic instructions spent in the unrolled loop that stores 64 bytes
- It’s is a `memset(…, 0, ...)` and can be replaced with CBO.ZERO
  - At least a 1.855% reduction in dynamic instructions...
- Valuable data for both software and hardware architects
Spotting trouble in the backend

Looking at the top contributors to dynamic instruction count helps spot worthwhile backend improvements.

- “Interesting” pattern of extensions around the minu
- Missed opportunities to use add.uw and sh2add.uw (following the minu)
- 35% reduction for this block, which will reduce the dynamic instruction count by 1.85%
Finding FIXMEs in ree.c

```c
/* Third, make sure the reaching definitions don't feed another and different extension. FIXME: this obviously can be improved. */
for (def = defs; def; def = def->next)
    if ((idx = def_map[INSN_UID (DF_REF_INSN (def->ref))])
        && idx != -1U
        && (cand = &(*insn_list)[idx - 1])
        && cand->code != code)
    {
        if (dump_file)
            {
            fprintf (dump_file, "Cannot eliminate extension:"
            print_rtl_single (dump_file, insn);
            fprintf (dump_file, " because of other extension\n"));
            }
        return;
    }
```
Next steps

Our “backlog” of things to work on

● Contribute the tools (especially the QEMU plug-in) back to the community
● Address the spotted code-generation issues
● Improve the analysis tools
  ○ Automate
  ○ Spot common problems
  ○ Compare individual benchmarking runs
● Add more benchmarks to improve coverage
● Run tests in additional configurations (e.g. RV32)
Community thoughts?

We encourage discussion on how we can make this into a useful tool to advance the RISC-V ecosystem:

- Is anyone else working on SPEC CPU 2017 performance
- Can we integrate this with GEM-5 and/or SPARTA
- How do we best share the infrastructure and jointly build analysis tools
  - Public hosting of results and co-existence with perf-results...
- What other benchmarks and workloads should be considered
- How to best collaborate
  - Avoid duplicating effort...
  - ...and share observations with actual micro-architectures between organisations.
- How to avoid useless work for the maintainers and get this committed