gcc-14: What's new and exciting for RISC-V

Bay Area RISC-V Meetup, Santa Clara, CA
May 23, 2024

Vineet Gupta, Rivos

Wait, why are we doing this?

Because why not :)

There seems to be a comprehensive resource out there already

https://gcc.gnu.org/gcc-14/changes.html

Well

"If a picture is worth a thousand words, a patch is worth at least a thousand pictures". [Will Deacon on lkml]

Share Interesting Technical tidbits in gcc-14

Share what Rivos Team has been up to in gcc software ecosystem

Drink some free Beer ;-)

Rivos

gcc-14 for RISC-V Highlights

Working Auto-Vectorization

- Mature enough to run entire SPEC CPU2017 (SPECint as well as SPECfp)

Support for ever growing list of New ISA extensions

- Zicond, Zfa to name a few

Improved code for Atomics

Pressure Sensitive Instruction Scheduling

Constant Synthesis Improvements

Initial Sign Extension Elimination Improvements

gcc-14 RISC-V updates: Auto-Vectorization Works

User writes

```
int abs(int i) {
  return i < 0 ? -i : i;
}
int sum_of_abs_diff (unsigned char *img1,
             unsigned char *img2, int n)
{
  int sum = 0;
  for (int i = 0; i < n; i++)
    sum += abs (img1[i] - img2[i]);
  return sum;
}
```

But Compiler needs to ...

```
for (int i = 0; i < N; i++) {
   // implicit type promotion
   int pix1 = (int) img1[i];
   int pix2 = (int) img2[i];
   // 32-bit sub (w/ sign-ext)
   int diff = pix1 - pix2;
   sum += abs (diff);
}
```

gcc 13.1
-O2 -march=rv64gcv_zbb -ftree-vectorize

```
abs:
        sraiw   a5,a0,31
        xor     a0,a0,a5
        subw    a0,a0,a5
        ret

sum_of_abs_diff:
        ble     a2,zero,.L6
        mv      a4,a0
        add     a2,a0,a2
        li      a0,0
.L5:
        lbu     a3,0(a1)
        lbu     a5,0(a4)
        addi    a4,a4,1
        addi    a1,a1,1
        subw    a5,a5,a3
        sraiw   a3,a5,31
        xor     a5,a3,a5
        subw    a5,a5,a3
        addw    a0,a5,a0
        bne     a2,a4,.L5
        ret
.L6:
        li      a0,0
        ret
```

gcc 14.1
-O2 -march=rv64gcv_zbb -ftree-vectorize

```
abs:
        negw    a5,a0
        max     a0,a5,a0
        ret

sum_of_abs_diff:
        ble     a2,zero,.L6
        vsetvli a5,zero,e32,m1,ta,ma
        vmv.v.i v2,0
.L5:
        vsetvli a5,a2,e8,mf4,ta,ma
        vle8.v  v3,0(a1)
        vle8.v  v4,0(a0)
        sub     a2,a2,a5
        add     a0,a0,a5
        add     a1,a1,a5
        vwsubu.vv       v1,v4,v3
        vsetvli zero,zero,e16,mf2,tu,mu
        vmv1r.v v3,v2
        vmslt.vi        v0,v1,0
        vneg.v  v1,v1,v0.t
        vwadd.wv        v2,v3,v1
        bne     a2,zero,.L5
        vsetvli a5,zero,e32,m1,ta,ma
        li      a4,0
        vmv.s.x v1,a4
        vredsum.vs      v2,v2,v1
        vmv.x.s a0,v2
        ret
.L6:
        li      a0,0
        ret
```

https://godbolt.org/z/6EhdsKjW8

gcc-14 RISC-V updates: Auto-Vectorization #2

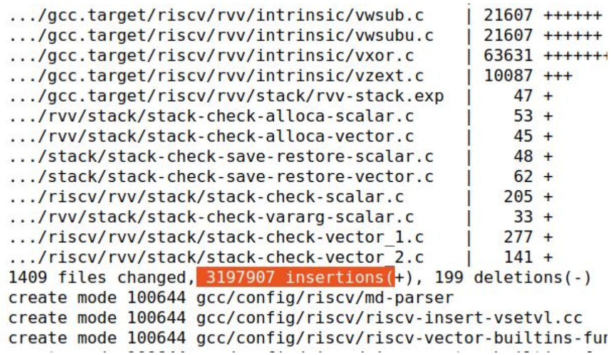
Both SLP and Loop Vectorizers enabled in the middle end

Length Agnostic (VLA) and Length Specific (VLS) supported

- Length Agnostic reads VLEN at runtime
- Stripmine style loops

Why all the fuss

- RVV ISA is special: Vector Insns predicated twice
  - VLEN mask and Regular v0 mask
- Needed new patterns in the middle-end:
  - WHILE_LEN
- Initial patch for RVV support was a mere ~3M lines in ~1400 files
  - https://gcc.gnu.org/pipermail/gcc-patches/2022-May/595903.html



```
.../gcc.target/riscv/rvv/intrinsic/vwsub.c    | 21607 ++++++
.../gcc.target/riscv/rvv/intrinsic/vwsubu.c   | 21607 ++++++
.../gcc.target/riscv/rvv/intrinsic/vxor.c     | 63631 +++++++
.../gcc.target/riscv/rvv/intrinsic/vzext.c    | 10087 +++
.../gcc.target/riscv/rvv/stack/rvv-stack.exp  |    47 +
.../rvv/stack/stack-check-alloca-scalar.c     |    53 +
.../rvv/stack/stack-check-alloca-vector.c     |    45 +
.../stack/stack-check-save-restore-scalar.c   |    48 +
.../stack/stack-check-save-restore-vector.c   |    62 +
.../riscv/rvv/stack/stack-check-scalar.c      |   205 +
.../rvv/stack/stack-check-vararg-scalar.c     |    33 +
.../riscv/rvv/stack/stack-check-vector_1.c    |   277 +
.../riscv/rvv/stack/stack-check-vector_2.c    |   141 +
1409 files changed, 3197907 insertions(+), 199 deletions(-)
create mode 100644 gcc/config/riscv/md-parser
create mode 100644 gcc/config/riscv/riscv-insert-vsetvl.cc
create mode 100644 gcc/config/riscv/riscv-vector-builtins-fur
```

Rivos

gcc-14 RISC-V updates: VSETVL Pass

Backbone of Auto-Vec in the RISC-V backend

- Compiler starts off by adding a VSETVL insn for each Vector insn
- A dedicated RISC-V Pass then uses standard Lazy Code Motion (LCM) algorithm to
  - Eliminate redundant VSETVL insns
  - Fuse compatible VSETVL insns and …
  - … Hoist them outside the loop

```
-O2 -march=rv64gcv
--param=vsetvl-strategy=simple

foo1:
        ble     a2,zero,.L5
        add     a2,a0,a2
.L3:
        vsetvli a5,zero,e8,mf8,ta,ma
        vle8.v  v1,0(a0)
        vsetvli a5,zero,e8,mf8,ta,ma
        addi    a0,a0,1
        vse8.v  v1,0(a1)
        addi    a1,a1,1
        bne     a0,a2,.L3
.L5:
        ret
```

```
-O2 -march=rv64gcv

foo1:
        ble     a2,zero,.L1
        add     a2,a0,a2
        vsetvli a5,zero,e8,mf8,ta,ma
.L3:
        vle8.v  v24,0(a0)
        addi    a0,a0,1
        vse8.v  v24,0(a1)
        addi    a1,a1,1
        bne     a0,a2,.L3
.L1:
        ret
```

https://godbolt.org/z/5a5GMc4fE

```
#include “riscv_vector.h”

void foo1 (void * restrict in, void * restrict out, int n)
{
  for (int i = 0; i < n; i++) {
      vuint8mf8_t v = *(vuint8mf8_t*)(in + i);
      *(vuint8mf8_t*)(out + i) = v;
  }
}
```

gcc-14 Updates: Constant Synthesis Improvements

SPEC2017 Deepsjeng dynamic icounts increased in gcc 13.1 (regression)

- Top blocks involve Large constants
- RV ISA doesn't allow efficient encoding of Large constants
- But compiler are not supposed to make it any worse
- Compiler strategy changed over times
  - gcc 12 used constant pools
  - gcc-13 stated synthesizing them but terrible codegen
  - gcc-14 relaxed register usage restriction and reuse same lo/hi 32-bit values

2023-05-09 c104ef4b5eb1 RISC-V: improve codegen for large constants with same 32-bit lo and hi parts [2]

2023-04-11 0530254413f8 riscv: relax splitter restrictions for creating pseudos

2023-03-01 7e52f4420ffb RISC-V: costs: miscomputed shiftadd_cost triggering synth_mult [PR/108987]

```
long long foo(void)
{
   return 0x0101010101010101ull;
}
```

```
gcc 12.1: -O2 -march=rv64gc
f:
        lui     a5,%hi(.LC0)
        ld      a0,%lo(.LC0)(a5)
        ret
.LC0:
        .dword  72340172838076673
```

https://godbolt.org/z/hTTabeY4e

```
gcc 13.1: -O2 -march=rv64gc
f:
        li      a0,0x101_000
        addi    a0,a0,0x101
        slli    a0,a0,16
        addi    a0,a0,0x101
        slli    a0,a0,16
        addi    a0,a0,0x101
        ret
```

```
gcc 14.1: -O2 -march=rv64gc
f:
        li      a5,0x101_0000
        addi    a5,a5,0x101
        slli    a0,a5,32
        add     a0,a0,a5
        ret
```

Rivos

gcc-14 Updates: Sign Extension Elimination Improv #1

Extraneous Sign Extension in RISC-V gcc codegen is a systematic problem.

ISA and the ABI are to blame, but again compilers can do better

- Built into most basic of ALU insns: ADDW/ADDIW
  - signed 32-bit addition and final sign-extension into 64-bit reg
- RISC-V ABI mandates that 32-bit values (on rv64) are held as sign extended values in 64-bit wide reg.
  - Simplifies calling convention
  - Allows 64-bit comparison insn to work on 32-bit values

Decent amount of work went in gcc-14, but lot more needs to done!

2023-04-28 1966741378d5 RISC-V: Eliminate redundant zero extension of minu/maxu operands

2023-06-07 99bfdb072e67 RISC-V: Eliminate extension after for *w instructions

2023-10-24 fb4e2c1648ea RISC-V: elide unnecessary sign extend when expanding cmp_and_jump

2023-10-16 8eb9cdd14218 expr: don't clear SUBREG_PROMOTED_VAR_P flag for a promoted subreg [target/111466]

Rivos

gcc-14 Updates: Sign Ext Elim Improv #2

Interesting tidbit with the last change in last slide

2023-10-16 8eb9cdd14218 expr: don't clear SUBREG_PROMOTED_VAR_P flag for a promoted subreg [target/111466]

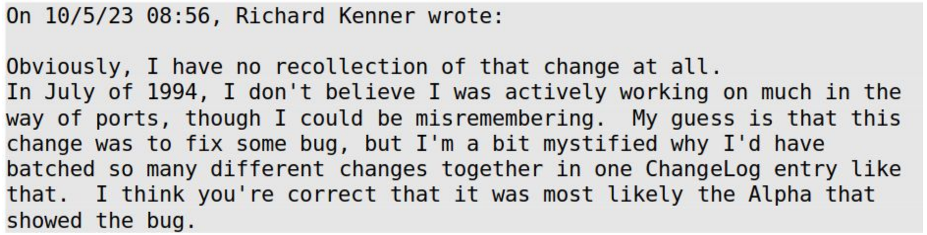
Just deletes a couple of lines

```
diff --git a/gcc/expr.cc b/gcc/expr.cc
index 308ddc09e631..d259c6e53385 100644
--- a/gcc/expr.cc
+++ b/gcc/expr.cc
@@ -9332,13 +9332,6 @@ expand_expr_real_2 (sepops ops, rtx target,
          op0 = expand_expr (treeop0, target, VOIDmode, modifier);
-         if (TYPE_UNSIGNED (TREE_TYPE (treeop0)) != unsignedp
-             && GET_CODE (op0) == SUBREG)
-           SUBREG_PROMOTED_VAR_P (op0) = 0;
-
          return REDUCE_BIT_FIELD (op0);
```

Added back in 1994 !

(with empty changelog)

1994-07-08 506980397227 (expand_expr, case CONVERT_EXPR): If changing signedness and we have a promoted SUBREG, clear the promotion flag.



```
On 10/5/23 08:56, Richard Kenner wrote:

Obviously, I have no recollection of that change at all.
In July of 1994, I don't believe I was actively working on much in the
way of ports, though I could be misremembering.  My guess is that this
change was to fix some bug, but I'm a bit mystified why I'd have
batched so many different changes together in one ChangeLog entry like
that.  I think you're correct that it was most likely the Alpha that
showed the bug.
```

gcc-14 Updates: Sign Ext Elim: Glimpse of work to be done

Redundant Extension Elimination REE Pass needs some love

- Can't handle "USE with more than One reaching DEFinition" or if they are in different BBs
- Can't leverage the ABI
    - Incoming "int" args are already sign-extended

```
/* PR/111467 */
int max(int a, int b)
{
   return (a > b) ? a : b;
}
```

gcc 14.1: -O2 -march=rv64gc

```
max(int, int):
        mv      a5,a0
        bge     a0,a1,.L4
        mv      a5,a1
.L4:
        sext.w  a0,a5
        ret
```

clang: -O2 -march=rv64gc

```
max(int, int):
        blt     a1, a0, .LBB1_2
        mv      a0, a1
.LBB1_2:
        ret
```

https://godbolt.org/z/qb3jvrnoe

gcc -fdump-rtl-ree

```
# DEF 1
(insn 8 4 11 2 (set (reg:SI 15 a5 [orig:137 b ] [137])
    (reg:SI 11 a1 [orig:136 b ] [136])) {*movsi_internal}

(jump_insn 11 8 22 2 (set (pc)
    (if_then_else (ge (reg/v:DI 11 a1 [orig:136 b ] [136])
            (reg/v:DI 10 a0 [orig:135 a ] [135]))
        (label_ref 13)
          (pc))) "max.c":12:20 273 {*branchdi}
          (int_list:REG_BR_PROB 536870916 (nil))
 -> 13)

# DEF 2
(insn 12 22 13 3 (set (reg:SI 15 a5 [orig:137 b ] [137])
    (reg:SI 10 a0 [orig:135 a ] [135])) {*movsi_internal}

(code_label 13 12 23 4 2 (nil) [1 uses])

# USE: Multiple reaching DEFs
(insn 19 14 20 4 (set (reg/i:DI 10 a0)
   sign_extend:DI (reg:SI 15 a5 [orig:137 b ] [137])))
        {extendsidi2}
```

gcc-14 updates: New pass f-m-o (**F**old **M**emory **O**ffsets)

> Why do fwprop or combine not what you want to do?

- Handles Partially Overlapping memory offset calculations.
- Handle multiple memory operations sharing some intermediate calculations.
- Handle late (post Reg Alloc) offset calculation (Stack pointer)

2023-10-16 04c9cf5c786b Implement new RTL optimizations pass: fold-mem-offsets

```
                     -ffold-mem-offsets

add t4, sp, 16       add t2, a6, sp
add t2, a6, t4
shl t3, t2, 1        shl t3, t2, 1
ld  a2, 0(t3)        ld  a2, 32(t3)    # 16 << 1
add a2, 1            add a2, 1
sd  a2, 8(t2)        sd  a2, 24(t2)    # 8 + 16
```

Rivos

Is gcc-14 worth it: how to quantify?

In an ideal world one could run linux perf on SPEC workloads and get aggregate cycle and instructions counts as well as statistical hotspots.

- Assumes working *perf tooling* on available *hardware* to execute *fast enough*.
- All three of which are kinda questionable in RISC-V ecosystem ATM.

Instead we measure dynamic icounts (vs. cycles) on Qemu user (vs. hardware) to run RISC-V binaries on an x86 host (fast enough).

- Kind of approximation but good first order
- Keeps development uarch agnostic and Vendor Neutral

Also this is a *SPEC centric world view*: one needs to start somewhere.

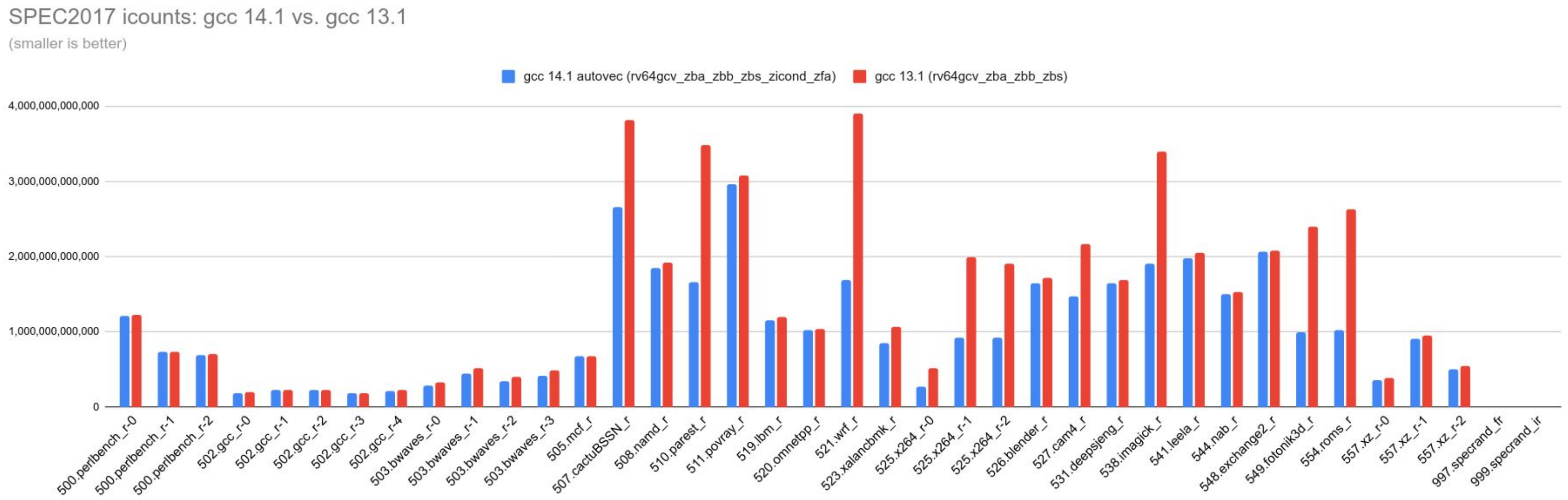
- To be clear this is not "best toggle for last perf drop drag race" (yet)
- The changes are fairly widely applicable to common workloads
    - Linux kernel build

SPEC dynamic icounts: RISC-V gcc-14 vs. gcc-13

Build: -Ofast -flto -static (no other toggle gimmicks)

Run: QEMU user with icount instrumentation plugin

Summary: 37,858,380,268,831 vs. 51,604,882,528,463 (~20% improvement, SMALLer is better)



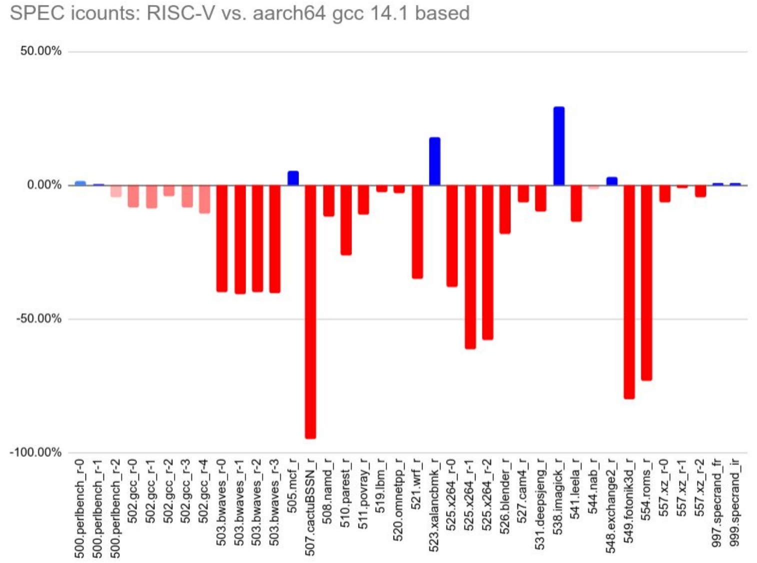
Rivos

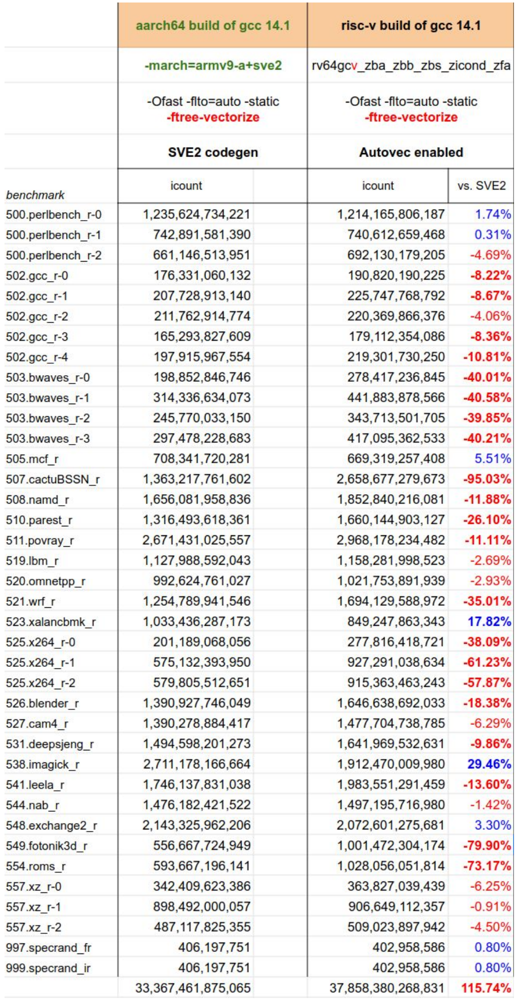
gcc-14 SPEC icounts: RISC-V vs. aarch64

Build: -Ofast -flto -static

- Exact same gcc-14.1 release sources based build - from scratch
- Numbers are for engineering, not marketing ;-)

Clearly much needs to be done





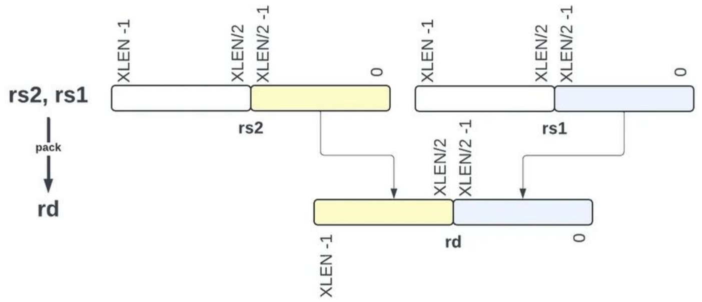
	aarch64 build of gcc 14.1	risc-v build of gcc 14.1	
	-march=armv9-a+sve2	rv64gcv_zba_zbb_zbs_zicond_zfa	
	-Ofast -flto=auto -static **-ftree-vectorize**	-Ofast -flto=auto -static **-ftree-vectorize**	
	SVE2 codegen	**Autovec enabled**	
benchmark	icount	icount	vs. SVE2
500.perlbench_r-0	1,235,624,734,221	1,214,165,806,187	1.74%
500.perlbench_r-1	742,891,581,390	740,612,659,468	0.31%
500.perlbench_r-2	661,146,513,951	692,130,179,205	-4.69%
502.gcc_r-0	176,331,060,132	190,820,190,225	**-8.22%**
502.gcc_r-1	207,728,913,140	225,747,768,792	**-8.67%**
502.gcc_r-2	211,762,914,774	220,369,866,376	-4.06%
502.gcc_r-3	165,293,827,609	179,112,354,086	**-8.36%**
502.gcc_r-4	197,915,967,554	219,301,730,250	**-10.81%**
503.bwaves_r-0	198,852,846,746	278,417,236,845	**-40.01%**
503.bwaves_r-1	314,336,634,073	441,883,878,566	**-40.58%**
503.bwaves_r-2	245,770,033,150	343,713,501,705	**-39.85%**
503.bwaves_r-3	297,478,228,683	417,095,362,533	**-40.21%**
505.mcf_r	708,341,720,281	669,319,257,408	5.51%
507.cactuBSSN_r	1,363,217,761,602	2,658,677,279,673	**-95.03%**
508.namd_r	1,656,081,958,836	1,852,840,216,081	**-11.88%**
510.parest_r	1,316,493,618,361	1,660,144,903,127	**-26.10%**
511.povray_r	2,671,431,025,557	2,968,178,234,482	**-11.11%**
519.lbm_r	1,127,988,592,043	1,158,281,998,523	-2.69%
520.omnetpp_r	992,624,761,027	1,021,753,891,939	-2.93%
521.wrf_r	1,254,789,941,546	1,694,129,588,972	**-35.01%**
523.xalancbmk_r	1,033,436,287,173	849,247,863,343	**17.82%**
525.x264_r-0	201,189,068,056	277,816,418,721	**-38.09%**
525.x264_r-1	575,132,393,950	927,291,038,634	**-61.23%**
525.x264_r-2	579,805,512,651	915,363,463,243	**-57.87%**
526.blender_r	1,390,927,746,049	1,646,638,692,033	**-18.38%**
527.cam4_r	1,390,278,884,417	1,477,704,738,785	-6.29%
531.deepsjeng_r	1,494,598,201,273	1,641,969,532,631	**-9.86%**
538.imagick_r	2,711,178,166,664	1,912,470,009,980	**29.46%**
541.leela_r	1,746,137,831,038	1,983,551,291,459	**-13.60%**
544.nab_r	1,476,182,421,522	1,497,195,716,980	-1.42%
548.exchange2_r	2,143,325,962,206	2,072,601,275,681	3.30%
549.fotonik3d_r	556,667,724,949	1,001,472,304,174	**-79.90%**
554.roms_r	593,667,196,141	1,028,056,051,814	**-73.17%**
557.xz_r-0	342,409,623,386	363,827,039,439	-6.25%
557.xz_r-1	898,492,000,057	906,649,112,357	-0.91%
557.xz_r-2	487,117,825,355	509,023,897,942	-4.50%
997.specrand_fr	406,197,751	402,958,586	0.80%
999.specrand_ir	406,197,751	402,958,586	0.80%
	33,367,461,875,065	37,858,380,268,831	**115.74%**

gcc-15 and beyond

Continued support for even more Extensions

- Zbkb - Scalar Crypto but has some Zbb like insns
  - PACK is really interesting for constant synthesis

https://fprox.substack.com/p/risc-v-scalar-bit-manipulation-extensions



New Pass for Enhanced Sign Extension Elimination

- Tracks register updates at byte, half-word and word-level

SATD support in the middle-end Auto-Vectorizer

Stack/Array Access codegen improvements

2024-03-06 9926c40a902e RISC-V: avoid LUI based const mat in alloca epilogue expansion

2024-05-13 f9cfc192ed01 RISC-V: avoid LUI based const mat in prologue/epilogue expansion [PR/105733]

2024-05-13 4bfc4585c993 RISC-V: avoid LUI based const materialization ... [part of PR/106265]

Eliding excessive Stack Spills during instruction scheduling

gcc @ Rivos: Community Engagement

Top 20 contributors in gcc-14 cycle

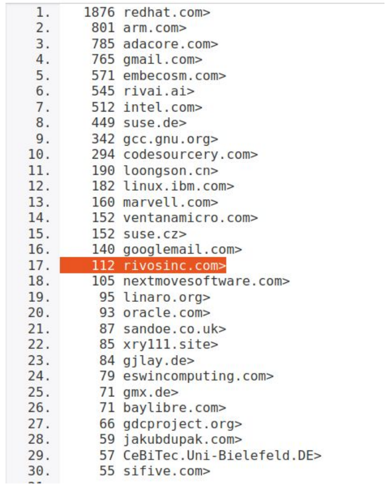
Very Active in the RISE Initiative

Setup public github CI for pre-commit (ewlu@rivosins.com) sanity check and post-commit (patrick@rivosinc.com).

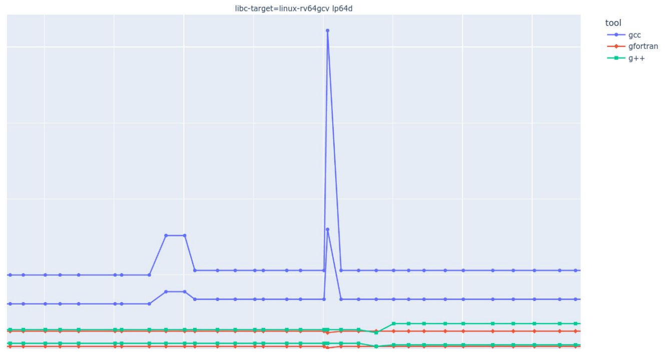
Setup Fuzzer to find latent bugs.

Detailed Analysis of RISC-V SPEC codegen quality vs. aarch64.

Helped drive RISE 3rd party contracts for further gcc improvements.



```
 1.   1876 redhat.com>
 2.    801 arm.com>
 3.    785 adacore.com>
 4.    765 gmail.com>
 5.    571 embecosm.com>
 6.    545 rivai.ai>
 7.    512 intel.com>
 8.    449 suse.de>
 9.    342 gcc.gnu.org>
10.    294 codesourcery.com>
11.    190 loongson.cn>
12.    182 linux.ibm.com>
13.    160 marvell.com>
14.    152 ventanamicro.com>
15.    152 suse.cz>
16.    140 googlemail.com>
17.    112 rivosinc.com>
18.    105 nextmovesoftware.com>
19.     95 linaro.org>
20.     93 oracle.com>
21.     87 sandoe.co.uk>
22.     85 xry111.site>
23.     84 gjlay.de>
24.     79 eswincomputing.com>
25.     71 gmx.de>
26.     71 baylibre.com>
27.     66 gdcproject.org>
28.     59 jakubdupak.com>
29.     57 CeBiTec.Uni-Bielefeld.DE>
30.     55 sifive.com>
```



Thank You !

vineetg@rivosinc.com

Rivos

Rivos gcc community Engagement: Competitive Analysis

Detailed analysis of RISC-V SPEC codegen (vs. aarch64)

- Truly apples-apples : nix based workflow to use exact same gcc sources to build gcc, build SPEC and run workloads in same test harness
- Helped drive RISE Technical proposals and RFP for 3rd party contractors for further gcc improvements
- Expedites gcc improvement needed in software ecosystem