Comparative Analysis of Compiler Performance for RISC-V on SPEC CPU 2017

Yongtai Li, Chunyu Liao, Ji Qiu
PLCT Lab. ISCAS
{liyongtai, chunyu, qiuji}@iscas.ac.cn

2025/3/1

Table of Contents

Background & Motivation

- RISC-V is growing fast in both embedded systems and high-performance computing. Code size is crucial for embedded systems, while dynamic instruction count matters a lot for HPC.
- SPEC CPU 2017, as an industry-standard benchmark, evaluates compiler performance across diverse workloads.
- Our goal is to analyze how LLVM and GCC perform in these aspects and identify potential improvements.

How We Tested

1. Setup
2. Data Collection
3. Automation

Setup

- Build GCC and LLVM on RISC-V hardware
- Build SPEC CPU 2017

Hardware: Milk-V Pioneer Box, 64 cores C920
Commit: GGC - d28ea8e5a704, LLVM - c9a6e993f7b3
Flags: `-Ofast` , `-flto` for C/C++, `-Ofast` for fortran

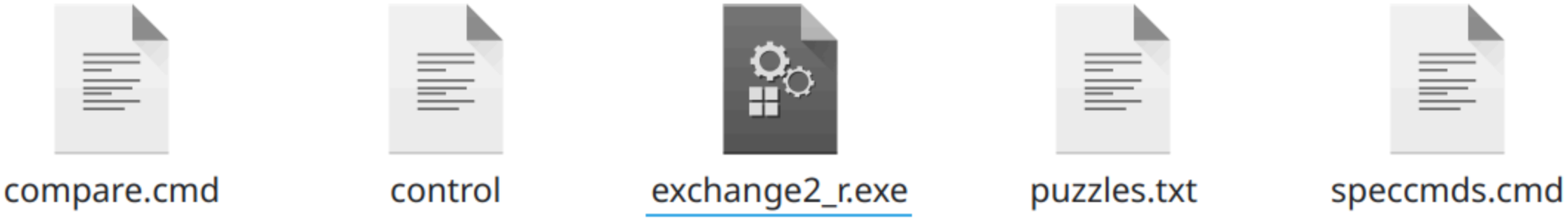
Targets: `rv64gbc` , `rv64gbcv`

Some tips：https://github.com/sihuan/llvm-work/tree/master/spec2017

Setup

Prepare the runtime environment, which includes input data and the `speccmds.cmd` file.

```
runcpu --config label.cfg --action runsetup intspeed
```



- `exchage2_r.exe` : Placeholder for the executable to be tested.
- `puzzles.txt` : Input data.
- `control` , `speccmds.cmd` , `compare.cmd` : Control files

Setup

We can use the `specinvoke` command to see how the tests run as described in `speccmds.cmd`

```
$   specinvoke -n speccmds.cmd
    # specinvoke r4356
    #  Invoked as: specinvoke -n speccmds.cmd
    # timer ticks over every 1000 ns
    # Use another -n on the command line to see chdir commands and env dump
    # Starting run for copy #0
    ../run_base_refrate_llvm-c9a6e993f7b3-rv64gc_zba_zbb_zbs-64.0000/\
    exchange2_r_base.llvm-c9a6e993f7b3-rv64gc_zba_zbb_zbs-64 6 > exchange2.txt 2>> exchange2.err
    specinvoke exit: rc=0
```

Data Collection

Code Size: strip binaries and measure their sizes.

DIC: Run tests using QEMU with the `insn` plugin.

```
$ path/to/qemu-riscv64 -plugin path/to/plugin/libinsn.so -d plugin ./demo
  cpu 0 insns: 20250301
  total insns: 20250301
```

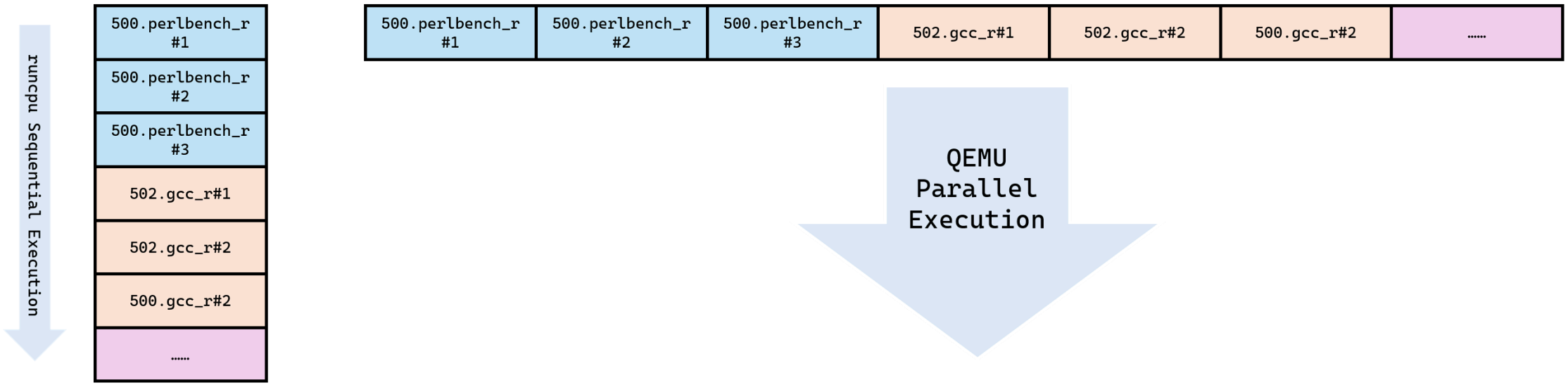
https://qemu-stsquad.readthedocs.io/en/latest/devel/tcg-plugins.html

https://github.com/qemu/qemu/blob/master/tests/tcg/plugins/insn.c

Automation

Now we can run any of the SPEC CPU benchmarks in QEMU and get the instruction count for it.
But such a process is tedious and inefficient, so we wrote an automated tool to handle this.
It has a web frontend that uploads a tarball containing several benchmark binaries, and then it can run these tests simultaneously using multiple QEMU processes

https://github.com/sihuan/countspec


runcpu Sequential Execution
500.perlbench_r #1
500.perlbench_r #2
500.perlbench_r #3
502.gcc_r#1
502.gcc_r#2
500.gcc_r#2
……
500.perlbench_r #1
500.perlbench_r #2
500.perlbench_r #3
502.gcc_r#1
502.gcc_r#2
500.gcc_r#2
……
QEMU Parallel Execution

gcc-d28ea8e5a704-rv64gcv_zba_zbb_zbs-523-623-510.tar.xz

ID: 17 Size: 1912404

2017 gcc V CXX3.15.0 fix 523 623 510

Benchmarks:

523.xalancbmk_r 623.xalancbmk_s 510.parest_r

Size: ref

NEW TASK DOWNLOAD DATA VIEW CONFIG INTRATE INTSPEED FPRATE FPSPEED CLEAR

gcc-d28ea8e5a704-rv64gc_zba_zbb_zbs-523-623-510.tar.xz

ID: 16 Size: 1863988

2017 gcc S CXX3.15.0 fix 523 623 510

Benchmarks:

523.xalancbmk_r 623.xalancbmk_s 510.parest_r

Size: ref

NEW TASK DOWNLOAD DATA VIEW CONFIG INTRATE INTSPEED FPRATE FPSPEED CLEAR

2006-gcc-d28ea8e5a704-rv64gcv_zba_zbb_zbs.tar.xz

ID: 15 Size: 14739172

2006 gcc v

Benchmarks:

400.perlbench 401.bzip2 403.gcc 410.bwaves 416.gamess 429.mcf 433.milc 434.zeusmp 435.gromacs 436.cactusADM 437.leslie3d 444.namd 445.gobmk 447.dealII 450.soplex 453.povray 454.calculix 456.hmmer 458.sjeng 459.GemsFDTD 462.libquantum 464.h264ref 465.tonto 470.lbm 471.omnetpp 473.astar 481.wrf 482.sphinx3

Size: ref

gcc-d28ea8e5a704-rv64gcv_zba_zbb_zbs.tar.xz

ID: 9 Size: 42189256

after add libgfortran

Benchmarks:

500.perlbench_r 502.gcc_r 503.bwaves_r 505.mcf_r 507.cactuBSSN_r 508.namd_r 510.parest_r 511.povray_r 519.lbm_r 520.omnetpp_r 521.wrf_r 523.xalancbmk_r 525.x264_r 526.blender_r 527.cam4_r 531.deepsjeng_r 538.imagick_r 541.leela_r 544.nab_r 548.exchange2_r 549.fotonik3d_r 554.roms_r 557.xz_r 600.perlbench_s 602.gcc_s 603.bwaves_s 605.mcf_s 607.cactuBSSN_s 619.lbm_s 620.omnetpp_s 621.wrf_s 623.xalancbmk_s 625.x264_s 627.cam4_s 628.pop2_s 631.deepsjeng_s 638.imagick_s 641.leela_s 644.nab_s 648.exchange2_s 649.fotonik3d_s 654.roms_s 657.xz_s

Size: ref

NEW TASK DOWNLOAD DATA VIEW CONFIG INTRATE INTSPEED FPRATE FPSPEED CLEAR

500.perlbench_r#1	instructions: 1215628841939	stdout: checkspam.2500.5.25.11.150.1.1.1.1.out	stderr: checkspam.2500.5.25.11.150.1.1.1.1.err
500.perlbench_r#2	instructions: 748347701632	stdout: diffmail.4.800.10.17.19.300.out	stderr: diffmail.4.800.10.17.19.300.err
500.perlbench_r#3	instructions: 704863956493	stdout: splitmail.6400.12.26.16.100.0.out	stderr: splitmail.6400.12.26.16.100.0.err
502.gcc_r#1	instructions: 197449405420	stdout: gcc-pp.opts-O3_-finline-limit_0_-fif-conversion_-fif-conversion2.out	stderr: gcc-pp.opts-O3_-finline-limit_0_-fif-conversion_-fif-conversion2.err
502.gcc_r#2	instructions: 234707404491	stdout: gcc-pp.opts-O2_-finline-limit_36000_-fpic.out	stderr: gcc-pp.opts-O2_-finline-limit_36000_-fpic.err
502.gcc_r#3	instructions: 227145124631	stdout: gcc-smaller.opts-O3_-fipa-	stderr: gcc-smaller.opts-O3_-fipa-

Code Size Comparison

Table 1: LLVM Code Size Relative to GCC (Scalar)

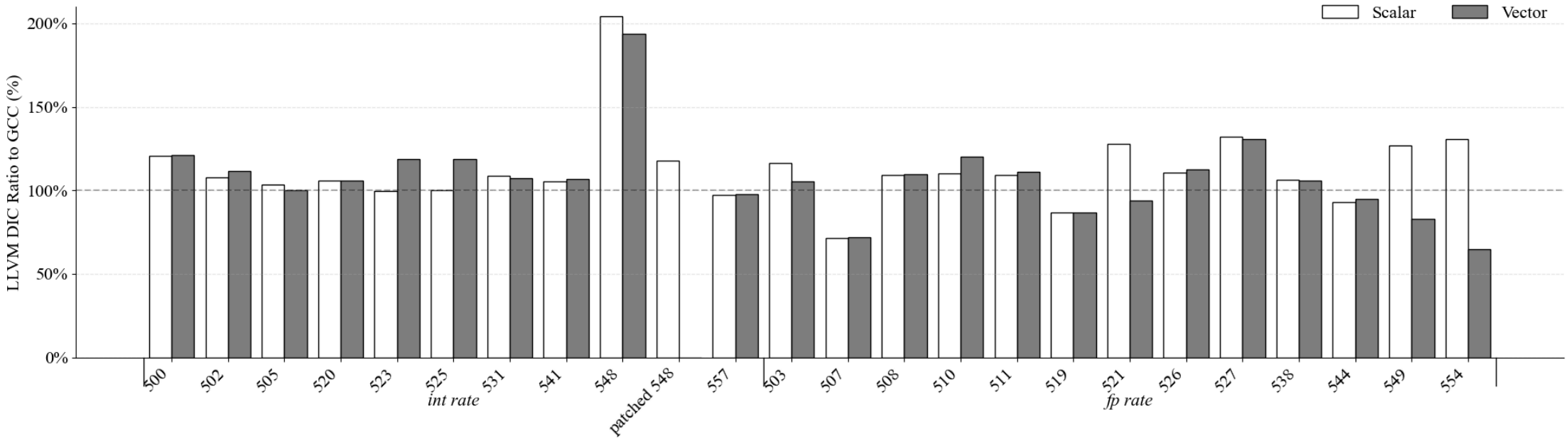
Range	C/C++	Fortran
< 0.9	508,541,507,519,531 557,538,525,505,544	527
0.9-1	500,502	
1-1.1	511,510,520	
> 1.1	523,526	521,554,549,548,503

Google Sheet QR Code

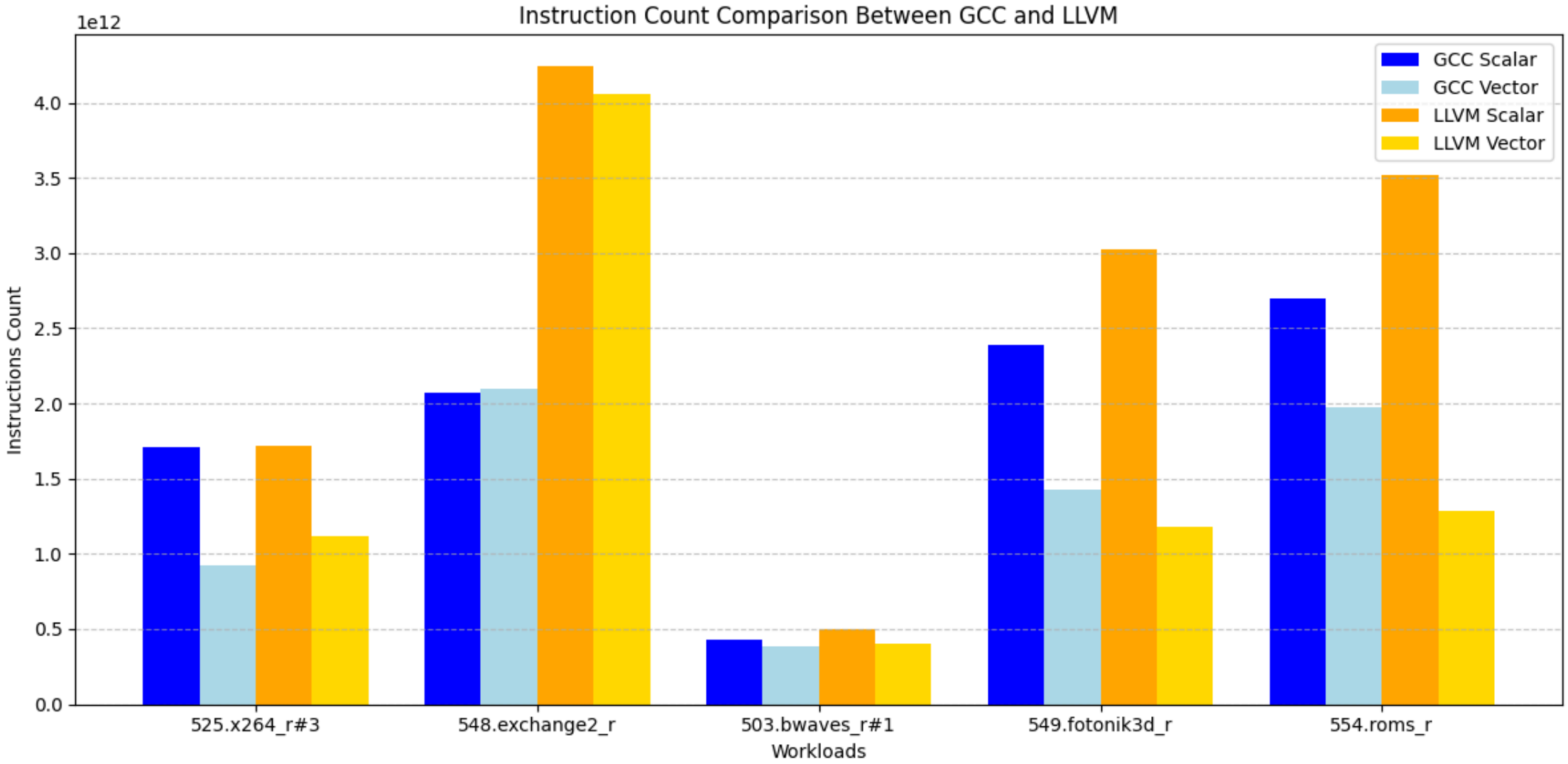
Benchmark	gcc s	gcc v	llvm s	llvm v	llvms/gccs	llvmv/gccv
508.namd_r	645960	658328	408984	426632	63.31%	64.81%
541.leela_r	105288	109464	80056	83656	76.04%	76.42%
527.cam4_r	18509560	19053760	11304944	11878504	61.08%	62.34%
519.lbm_r	18864	18952	14808	15088	78.50%	79.61%
531.deepsjeng_r	76120	80296	60992	79288	80.13%	98.74%
557.xz_r	137544	141736	113904	119952	82.81%	84.63%
538.imagick_r	1400704	1429464	1195560	1206960	85.35%	84.43%
525.x264_r	1015032	1076800	883728	936640	87.06%	86.98%
505.mcf_r	22808	22896	19888	21144	87.20%	92.35%
544.nab_r	91712	91808	82272	83744	89.71%	91.22%
500.perlbench_r	2276592	2309456	2062088	2075928	90.58%	89.89%
502.gcc_r	9562056	9635904	9071992	9368568	94.87%	97.23%
511.povray_r	1013680	1038416	1044176	1075616	103.01%	103.58%
510.parest_r	1514696	1599968	1625808	1740664	107.34%	108.79%
520.omnetpp_r	1633032	1649488	1767824	1774944	108.25%	107.61%
523.xalancbmk_r	3156704	3226480	3667744	3726384	116.19%	115.49%
526.blender_r	14898976	15154768	17502152	17710776	117.47%	116.87%

Benchmark	gcc s	gcc v	llvm s	llvm v	llvms/gccs	llvmv/gccv
527.cam4_r	18509560	19053760	11304944	11878504	61.08%	62.34%
521.wrf_r	30947416	35897608	41985072	48908776	135.67%	136.25%
554.roms_r	840072	1311384	2109832	2393064	251.15%	182.48%
549.fotonik3d_r	299200	326704	891232	957944	297.87%	293.21%
548.exchange2_r	127456	147800	1428760	1464288	1120.98%	990.72%
503.bwaves_r	27136	44360	549880	560992	2026.39%	1264.63%

Dynamic Instruction Count



Dynamic Instruction Count



Dynamic Instruction Count

Table 2: V-Ext DIC Reduction

Suit	GCC	LLVM
int-rate avg.	6.45%	4.22%
fp-rate avg.	11.73%	16.84%
all avg.	9.43%	11.35%

A case study on 548_exchange

The 548.exchange2_r benchmark is a Sudoku solver for 9×9 grids, written in Fortran 95 with approximately 1,600 lines of code. The program heavily relies on recursion, with a maximum recursion depth of up to 8 levels. Notably, it does not perform any floating-point operations, focusing entirely on integer computations.

The difference between LLVM and GCC is significant, regardless of whether the V extension is enabled.

In fact, this issue is not related to the B extension, nor is it even specific to the RISC-V architecture.

https://www.spec.org/cpu2017/Docs/benchmarks/548.exchange2_r.h

Some trouble

- Plan to use the perf tool, which needs to run on a physical machine. However, our RISC-V processor does not support the B extension or the V extension.
    - Reproduced this issue on rv64gc first.
- How to manually compile this 548_exchange benchmark?

```
flang-new -c -o exchange2.fppized.o -march=rv64gc -Ofast exchange2.fppized.f90
flang-new -march=rv64gc -Ofast exchange2.fppized.o -o exchange2_r
```

- How to manually run the tests?

```
./exchange2_r 0 # test size, solve the first problem in `puzzles.txt`
./exchange2_r 6 # ref size, solve all the six problems in `puzzles.txt`
```

We used `perf` to record some data for test size tests:

```
perf stat ./exchange2_r 0
perf report
```

GCC:

```
# Overhead  Command      Shared Object                Symbol
    38.96%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_digits_2.constprop.4.isra.0
    19.95%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_digits_2.constprop.3.isra.0
     9.03%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_digits_2.constprop.6.isra.0
     7.10%  exchange2_r  libgfortran.so.5.0.0         [.] _gfortran_mminloc0_4_i4
     6.25%  exchange2_r  exchange2_r                  [.] __logic_MOD_new_solver
     5.50%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_digits_2.constprop.5.isra.0
     4.41%  exchange2_r  exchange2_r                  [.] specific.4
     2.15%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_digits_2.constprop.7.isra.0
     2.07%  exchange2_r  exchange2_r                  [.]
$xrv64i2p1_m2p0_a2p1_f2p2_d2p2_c2p0_zicsr2p0_zifencei2p0_zmmul1p0
     1.21%  exchange2_r  exchange2_r                  [.] hidden_pairs.2
     0.85%  exchange2_r  exchange2_r                  [.] naked_triplets.1
     0.71%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_brute
     0.64%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_digits_2.constprop.2.isra.0
     0.33%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_digits_2.constprop.1.isra.0
     0.19%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_covered.constprop.0.isra.0
     0.12%  exchange2_r  exchange2_r                  [.] __brute_force_MOD_rearrange.isra.0
```

LLVM:

```
# Overhead  Command      Shared Object                Symbol
    88.78%  exchange2_r  exchange2_r                  [.] _QMbrute_forcePdigits_2
     5.78%  exchange2_r  exchange2_r                  [.] _QMlogicFnew_solverPspecific
     1.34%  exchange2_r  exchange2_r                  [.] _QMlogicPnew_solver
     0.67%  exchange2_r  exchange2_r                  [.] _QMlogicFnew_solverPhidden_triplets
     0.33%  exchange2_r  exchange2_r                  [.] _QMlogicFnew_solverPhidden_pairs
     0.28%  exchange2_r  exchange2_r                  [.] _QMlogicFnew_solverPnaked_triplets
     0.28%  exchange2_r  exchange2_r                  [.] _QMlogicFnew_solverPnaked_pairs
     0.20%  exchange2_r  exchange2_r                  [.] Fortran::runtime::Assign
     0.19%  exchange2_r  exchange2_r                  [.] _QMbrute_forcePbrute
     0.16%  exchange2_r  exchange2_r                  [.] Fortran::runtime::ReduceDimToScalar<int,
Fortran::runtime::ExtremumLocAccumulator<Fortran::runtime::NumericCompare<int, true, false> > >
     0.15%  exchange2_r  libc.so.6                    [.] memcpy
     0.15%  exchange2_r  libc.so.6                    [.] memset
     0.15%  exchange2_r  exchange2_r                  [.] _QMlogicFnew_solverPx_wing
     0.12%  exchange2_r  libc.so.6                    [.] _int_free
     0.11%  exchange2_r  libc.so.6                    [.] malloc
     0.10%  exchange2_r  exchange2_r                  [.] _QMbrute_forcePcovered
```

GCC:

```
$ sudo perf stat -B -e cache-references,cache-misses,cycles,instructions,branches,faults,migrations ./exchange2_r 6
  Performance counter stats for './exchange2_r 6':
       417,772,422      cache-references
         4,425,518      cache-misses                     #    1.059 % of all cache refs
 1,327,980,494,790      cycles
 2,081,545,960,539      instructions                     #    1.57  insn per cycle
   274,477,603,251      branches
                90      faults
                 0      migrations
     664.238029637 seconds time elapsed
     664.003383000 seconds user
       0.015994000 seconds sys
```

LLVM:

```
$ sudo perf stat -B -e cache-references,cache-misses,cycles,instructions,branches,faults,migrations ./exchange2_r 6
  Performance counter stats for './exchange2_r 6':
       656,853,453      cache-references
         6,735,733      cache-misses                     #    1.025 % of all cache refs
 2,373,045,060,474      cycles
 4,328,214,832,776      instructions                     #    1.82  insn per cycle
   496,851,214,471      branches
                83      faults
                 0      migrations
    1186.975610235 seconds time elapsed
    1186.570645000 seconds user
       0.007997000 seconds sys
```

Disassembly analysis

GCC:

```
$ objdump --disassemble=__brute_force_MOD_digits_2.isra.0 exchange2_r | wc -l
2312
$ for i in {1..7}; do objdump --disassemble=__brute_force_MOD_digits_2.constprop.${i}.isra.0 exchange2_r | wc -l; done
1094
922
1094
1145
752
925
1025
```

LLVM:

```
$ objdump --disassemble=_QMbrute_forcePdigits_2 exchange2_r | wc -l
2794
```

Based on the preliminary analysis of `perf`, the `digits_2` function in the GCC version has been split into forms like `__brute_force_MOD_digits_2.constprop.${1-7}.isra.0`, and the static assembly code lines of these functions are much smaller than those in LLVM.

So WHY?

In GCC, the hotspot function `digits_2` is split into several specialized versions. This specialization is caused by interprocedural constant propagation optimization (IPA-CP). One of the main effects of this optimization is the elimination of conditional branches.

Therefore, the assembly line count for each specialized version of the function is smaller.

The corresponding optimization pass in LLVM is `IPSCCP` Pass.

Verification

Disable this optimization in GCC by add the `-fno-ipa-cp` parameter

	gcc -fno-ipa-cp	gcc
exchange2_r 0	93,554,141,493	55,981,214,885

The number of instructions has almost doubled!

Manually add this optimization in LLVM

```
flang-new -c -emit-llvm -o exchange2.fppized.ll -march=rv64gc -Ofast exchange2.fppized.f90
opt -passes="ipsccp" exchange2.fppized.ll -o exchange2.fppized.ipsccp.ll
flang-new -march=rv64gc -Ofast fppized.ipsccp.ll -o exchange2_r
```

	llvm	llvm + ipsccp
exchange2_r 0	114,450,486,604	70,380,347,586

The number of instructions has decreased, and through disassembly, it was found that `digits_2` was also split into something like `_QMbrute_forcePdigits_2.specialized.3`.

llvm-project / llvm / lib / Passes / **PassBuilderPipelines.cpp**

Code | Blame | 2189 lines (1824 loc) · 90.9 KB

```
402  PassBuilder::buildO1FunctionSimplificationPipeline(OptimizationLevel Level,
1116   if (Phase == ThinOrFullLTOPhase::ThinLTOPostLink)
1117     MPM.addPass(LowerTypeTestsPass(nullptr, nullptr, true));
1118
1119   invokePipelineEarlySimplificationEPCallbacks(MPM, Level);
1120
1121   // Interprocedural constant propagation now that basic cleanup has occurred
1122   // and prior to optimizing globals.
1123   // FIXME: This position in the pipeline hasn't been carefully considered in
1124   // years, it should be re-analyzed.
1125   MPM.addPass(IPSCCPPass(
1126               IPSCCPOptions(/*AllowFuncSpec=*/
1127                             Level != OptimizationLevel::Os &&
1128                             Level != OptimizationLevel::Oz &&
1129                             !isLTOPreLink(Phase))));
1130
1131   // Attach metadata to indirect call sites indicating the set of functions
1132   // they may target at run-time. This should follow IPSCCP.
1133   MPM.addPass(CalledValuePropagationPass());
1134
```

6 ■■■■■ llvm/lib/Passes/PassBuilderPipelines.cpp

```
@@ -1204,6 +1204,12 @@ PassBuilder::buildModuleSimplificationPipeline(OptimizationLevel Level,
1204 1204     else
1205 1205       MPM.addPass(buildInlinerPipeline(Level, Phase));
1206 1206
      1207 +   MPM.addPass(IPSCCPPass(
      1208 +               IPSCCPOptions(/*AllowFuncSpec=*/
      1209 +                             Level != OptimizationLevel::Os &&
      1210 +                             Level != OptimizationLevel::Oz &&
      1211 +                             !isLTOPreLink(Phase))));
      1212 +
1207 1213     // Remove any dead arguments exposed by cleanups, constant folding globals,
1208 1214     // and argument promotion.
1209 1215     MPM.addPass(DeadArgumentEliminationPass());
```

After reading the LLVM source code, we found that the `IPSCCP` Pass is enabled by default, and our previous manual run was effectively a repetition. After some attempts, we found an appropriate place to run the Pass again.

```
$ objdump -D exchange2_r_patched_llvm | grep "digits_2.*:$"
0000000000011ab0 <_QMbrute_forcePdigits_2>:
0000000000018a4e <_QMbrute_forcePdigits_2.specialized.1>:
0000000000019820 <_QMbrute_forcePdigits_2.specialized.2>:
000000000001a436 <_QMbrute_forcePdigits_2.specialized.3>:
000000000001ae78 <_QMbrute_forcePdigits_2.specialized.4>:
000000000001ba8e <_QMbrute_forcePdigits_2.specialized.5>:
000000000001c7e6 <_QMbrute_forcePdigits_2.specialized.6>:
000000000001d072 <_QMbrute_forcePdigits_2.specialized.7>:
000000000001dad0 <_QMbrute_forcePdigits_2.specialized.8>:
```

With this patch, LLVM now exhibits similar behavior, resulting in a substantial performance uplift.

Used `perf` again to obtain the instruction count for `exchange2_r 0` on rv64gc, as shown in the table right.

Additionally, on x86_64, it has a similar result.

Compiler	Instructions on rv64gc
GCC #d28ea8e5	55,965,728,914
LLVM #62d44fbd	105,416,890,241
LLVM #62d44fbd with patch	62,693,427,761

Compiler	cpu_atom instructions on x86_64
LLVM #62d44fbd	100,147,914,793
LLVM #62d44fbd with patch	53,077,337,115

Conclusion & Future Work

- LLVM and new flang are ready for real-world workloads on RISC-V.
- LLVM produces smaller C/C++ binaries but struggles with Fortran.
- GCC is better at reducing dynamic instruction count in integer workloads.
- LLVM’s auto-vectorization for floating-point workloads is ahead of GCC.

In the future, we will contunie the work to reduce the prefmance difference between GCC and LLVM.

Resources

Code Size data: https://docs.google.com/spreadsheets/d/1e6sAkT1kZa8LQo4MWgT-NomF8fSHnClrJMVTrxktUAM

DIC data: https://docs.google.com/spreadsheets/d/1BSSc5XRr_QUmEgupRs3MgUJ4pICWsNW_X25vADO7DBY

countspec: https://github.com/sihuan/countspec

Workaround for 548: https://github.com/llvm/llvm-project/pull/96620

email: liyongtai@iscas.ac.cn

Thanks