# LLVM vs. GCC on RISC-V Using SPEC CPU Benchmarks: Methods, Gaps, and Optimizations

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

2025/6/10

# 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, 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.

# Methods

## How We Tested

1. Build Compilers & Benchmarks
2. Run and collect data
3. Automation

# Build Compilers & Benchmarks

Hardware: Milk-V Pioneer Box, 64 cores C920

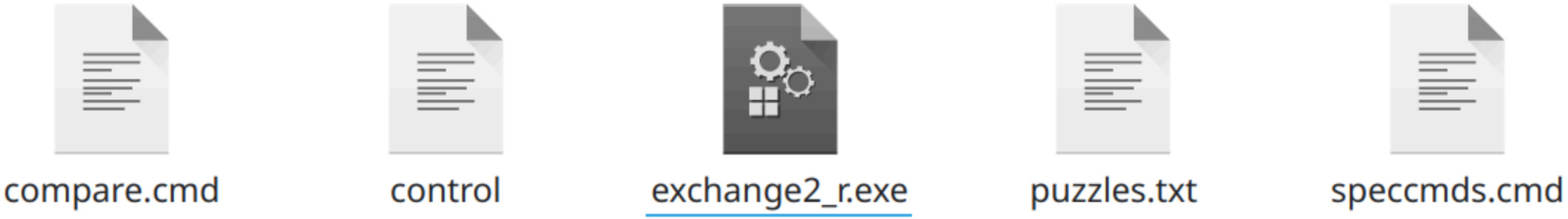
Commit: GGC - d28ea8e5a704, LLVM - c9a6e993f7b3

Flags: `-Ofast` , `-flto` for C/C++, `-Ofast` for fortran

Targets: `rv64gc_zba_zbb_zbs` , `rv64gcv_zba_zbb_zbs`

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.
- `speccmds.cmd` , `compare.cmd` : Control files

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.

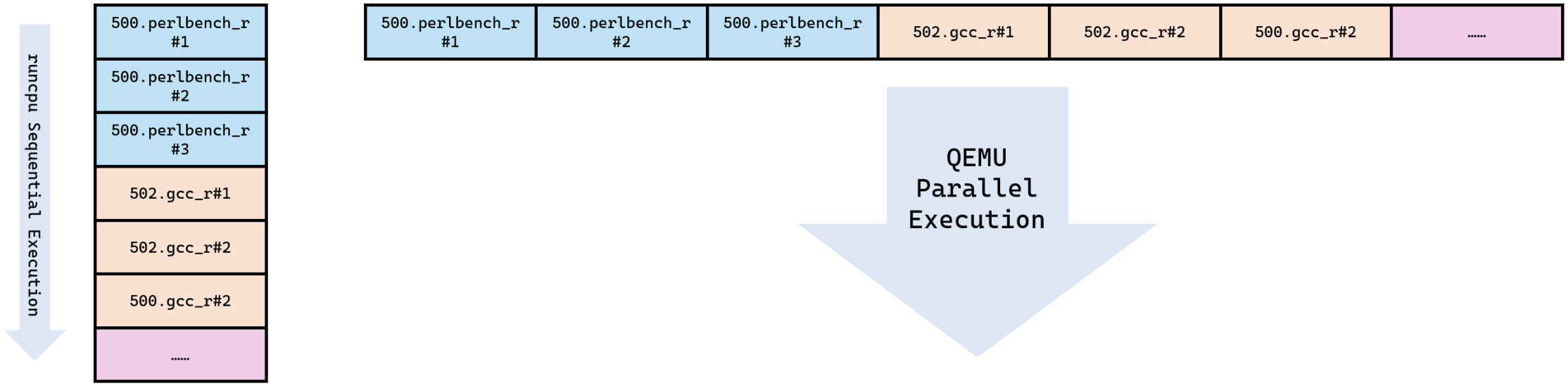
```
$ path/to/qemu-riscv64 -plugin path/to/plugin/libinsn.so -d plugin ./demo
  cpu 0 insns: 20250610
  total insns: 20250610
```

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

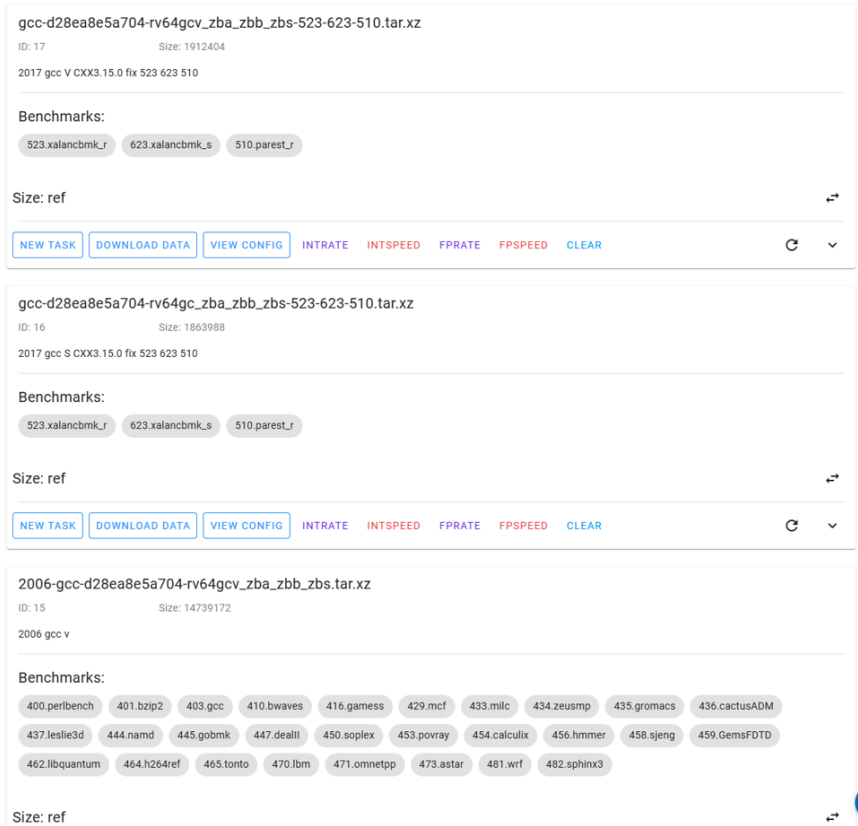
https://github.com/qemu/qemu/blob/master/tests/tcg/plugins/insn.c

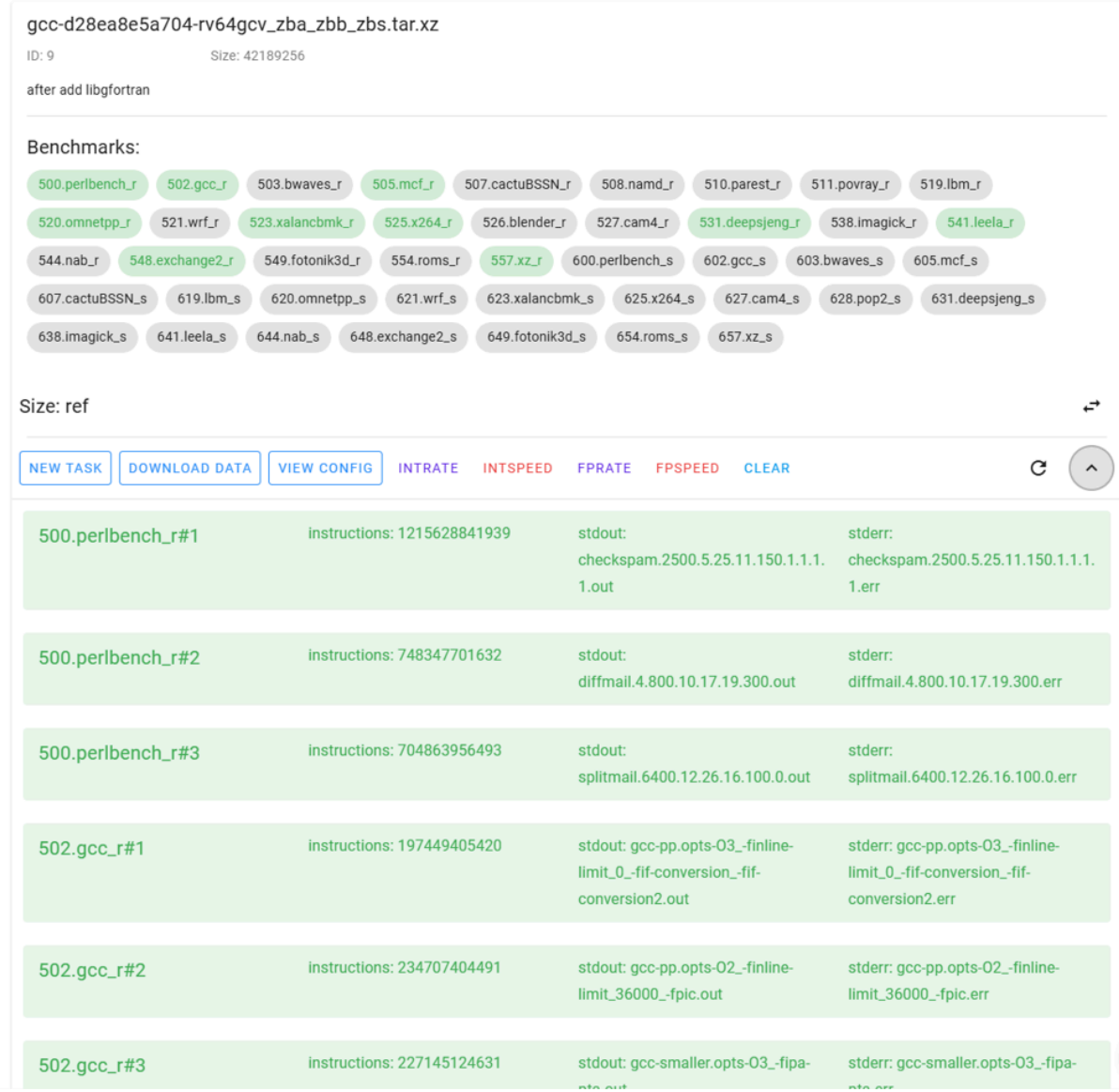
# Automation

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


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)**

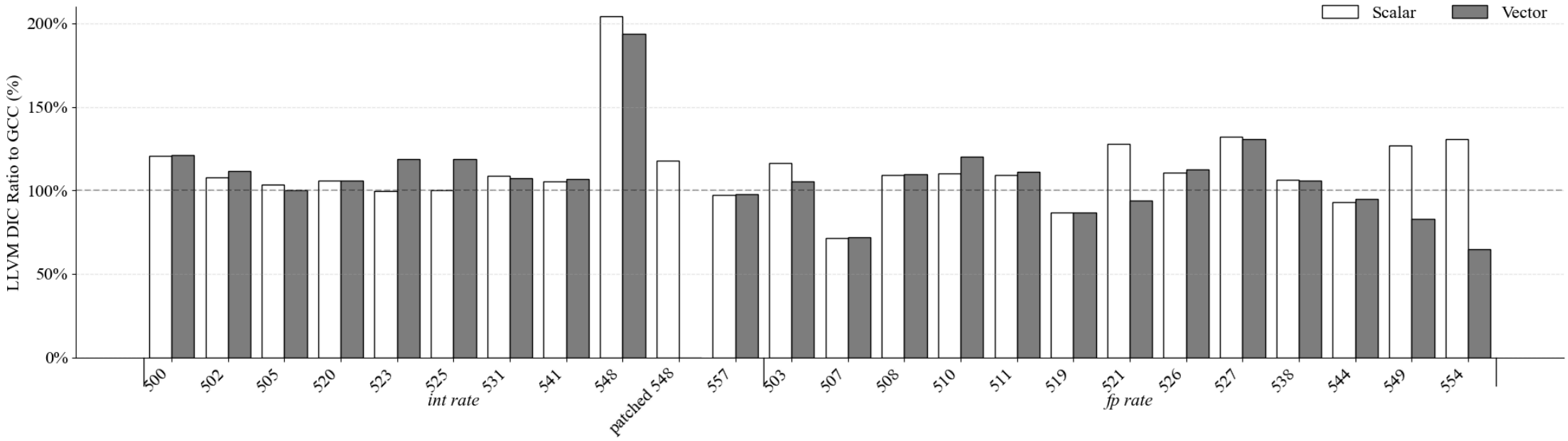
| Range | C/C++ | Fortran |
|---|---|---|
| < 0.9 | 508,541,507,519,531<br>557,538,525,505,544 | 527 |
| 0.9-1 | 500,502 | |
| 1-1.1 | 511,510,520 | |
| > 1.1 | 523,526 | 521,554,549,548,503 |

Google Sheet QR Code

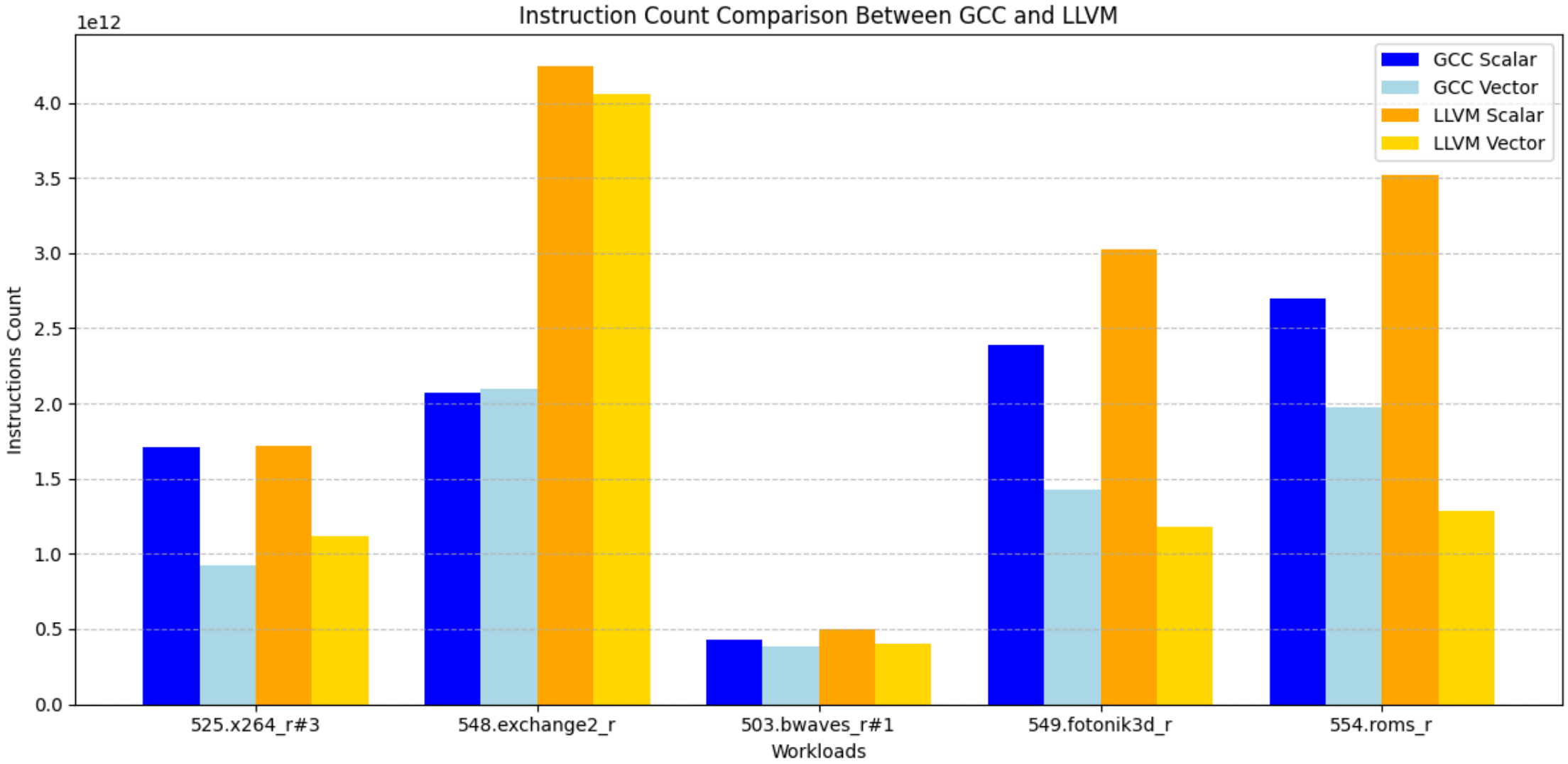
| Benchmark | gcc s | gcc v | llvm s | llvm v | llvms/gccs | llvmv/gccv |
|---|---|---|---|---|---|---|
| 508.namd_r | 645960 | 658328 | 408984 | 426632 | 63.31% | 64.81% |
| 541.leela_r | 105288 | 109464 | 80056 | 83656 | 76.04% | 76.42% |
| 527.cam4_r | 18509560 | 19053760 | 11304944 | 11878504 | 61.08% | 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% |

# Optimizations

## 548_exchange_r

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 performance difference between GCC and LLVM is significant: LLVM executes approximately twice as many dynamic instructions. This trend holds consistently across both `x86` and `ARM` architectures.

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

Subsequent experiments were conducted exclusively on the `RV64GC` platform, primarily utilizing the `objdump` and `perf` tools for analysis.

- 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:

```
$ 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.

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.

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

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

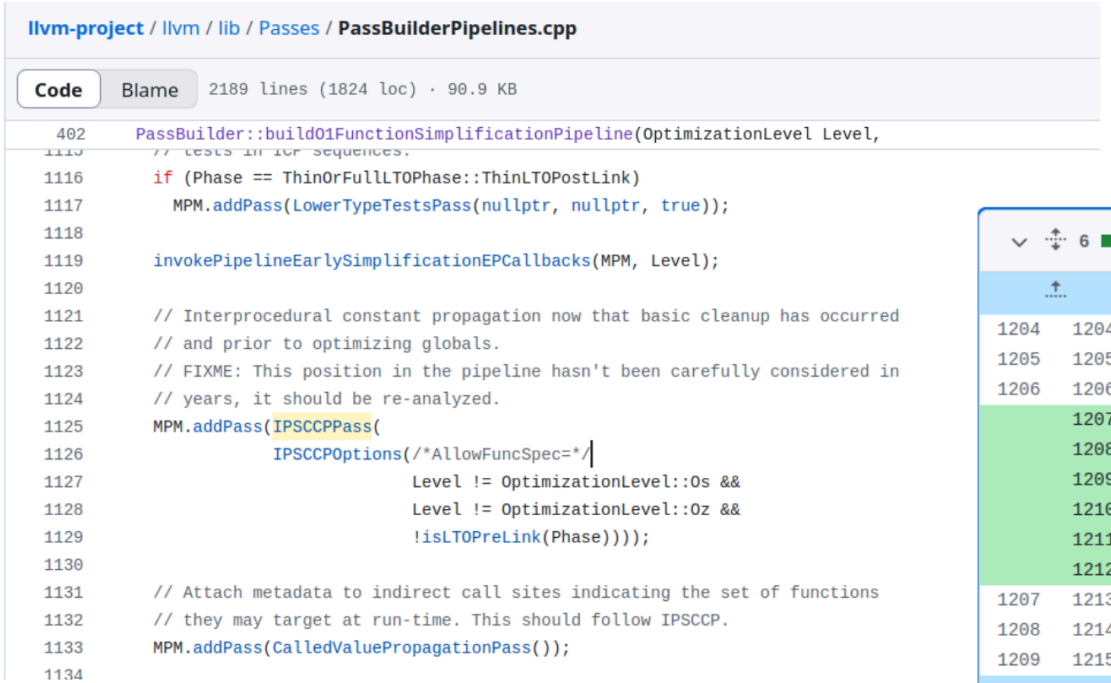
The number of instructions has almost doubled!

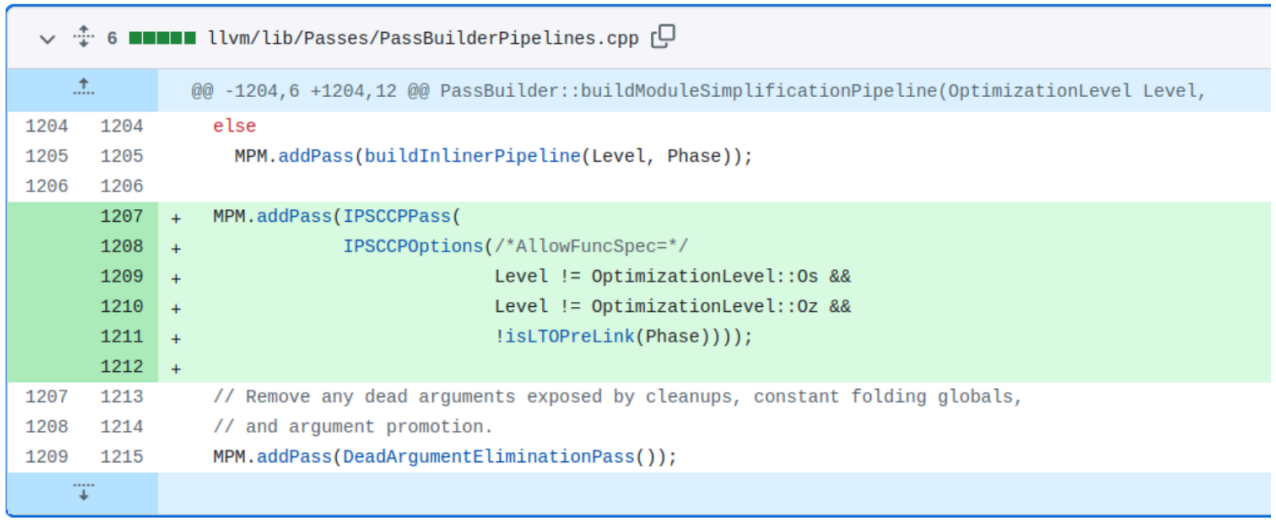
## Manually running an extra IPSCCP Pass

```
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 with extra 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`.




Repeatedly running the `IPSCCP` pass after the inliner pipeline will be effective.

```
$ 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>:
```

| 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 |

# BUT

- Running IPSCCP twice causes a massive compile-time regression
- Simply running Argument Promotion before IPSCCP is enough

https://github.com/llvm/llvm-project/pull/111163

## 473.astar

471.astar (pronounced: A-star) is derived from a portable 2D path-finding library that is used in game's AI. This library implements three different path-finding algorithms: First is the well known A* algorithm for maps with passable and non-passable terrain types. Second is a modification of the A* path finding algorithm for maps with different terrain types and different move speed. Third is an implementation of A* algorithm for graphs. This is formed by map regions with neighborhood relationship. The library also includes pseudo-intellectual functions for map region determination.

https://www.spec.org/cpu2006/Docs/473.astar.html

**perf report，** GCC ↓ ——————————————————— LLVM ↓

Samples: 58K of event 'cycles', Event count (approx.): 28022698026

| Overhead | Command | Shared Object | Symbol |
|---|---|---|---|
| 40.99% | astar | astar | [.] wayobj::makebound2 |
| 30.06% | astar | astar | [.] regwayobj::makebound2 |
| 15.51% | astar | astar | [.] way2obj::releasepoint |
| 7.76% | astar | astar | [.] regmngobj::getregfillnum |
| 1.92% | astar | astar | [.] way2obj::fill |
| 1.50% | astar | astar | [.] regwayobj::getway |
| 0.31% | astar | astar | [.] wayobj::createwayar |
| 0.23% | astar | astar | [.] regboundobj::makebound2 |
| 0.21% | astar | astar | [.] regmngobj::defineneighborh |
| 0.17% | astar | libc.so.6 | [.] malloc |
| 0.13% | astar | astar | [.] regmngobj::makebound2 |
| 0.12% | astar | libc.so.6 | [.] _int_free |
| 0.12% | astar | astar | [.] wayobj::fill |
| 0.11% | astar | astar | [.] regwayobj::fill |
| 0.08% | astar | astar | [.] main |
| 0.07% | astar | libc.so.6 | [.] _int_malloc |
| 0.07% | astar | astar | [.] way2obj::createwayar |
| 0.06% | astar | astar | [.] wayobj::makeobstaclebound2 |
| 0.05% | astar | libc.so.6 | [.] free |
| 0.03% | astar | astar | [.] regboundobj::firststep |
| 0.03% | astar | astar | [.] regobj::create |
| 0.03% | astar | [kernel.kallsyms] | [k] memset |
| 0.03% | astar | astar | [.] regmngobj::createregions |
| 0.03% | astar | astar | [.] regmngobj::enlargeneighbor |
| 0.02% | astar | astar | [.] random1 |
| 0.02% | astar | libc.so.6 | [.] unlink_chunk.constprop.0 |
| 0.02% | astar | astar | [.] regmngobj::definemiddlereg |
| 0.02% | astar | astar | [.] regmngobj::findfreeplace |
| 0.02% | astar | astar | [.] regboundobj::step |
| 0.02% | astar | [kernel.kallsyms] | [k] _raw_spin_unlock_irqrestor |
| 0.01% | astar | astar | [.] myrandom |
| 0.01% | astar | [kernel.kallsyms] | [k] change_protection |
| 0.01% | astar | libc.so.6 | [.] _wordcopy_fwd_aligned |
| 0.01% | astar | astar | [.] regmngobj::addregions |

Samples: 66K of event 'cycles', Event count (approx.): 32218702930

| Overhead | Command | Shared Object | Symbol |
|---|---|---|---|
| 37.45% | astar | astar | [.] wayobj::makebound2 |
| 22.86% | astar | astar | [.] way2obj::releasepoint |
| 18.68% | astar | astar | [.] regwayobj::makebound2 |
| 9.14% | astar | astar | [.] regwayobj::isaddtobound |
| 6.35% | astar | astar | [.] regmngobj::getregfillnum |
| 2.18% | astar | astar | [.] way2obj::fill |
| 1.22% | astar | astar | [.] regwayobj::getway |
| 0.44% | astar | astar | [.] wayobj::createwayar |
| 0.21% | astar | astar | [.] regboundobj::makebound2 |
| 0.13% | astar | astar | [.] regmngobj::defineneighbor |
| 0.12% | astar | astar | [.] regwayobj::fill |
| 0.12% | astar | libc.so.6 | [.] malloc |
| 0.11% | astar | astar | [.] wayobj::fill |
| 0.11% | astar | astar | [.] regmngobj::makebound2 |
| 0.09% | astar | libc.so.6 | [.] _int_free |
| 0.09% | astar | astar | [.] main |
| 0.06% | astar | astar | [.] way2obj::createwayar |
| 0.05% | astar | libc.so.6 | [.] free |
| 0.05% | astar | libc.so.6 | [.] _int_malloc |
| 0.04% | astar | astar | [.] wayobj::makeobstaclebound |
| 0.04% | astar | astar | [.] regboundobj::firststep |
| 0.03% | astar | astar | [.] random1 |
| 0.03% | astar | libc.so.6 | [.] unlink_chunk.constprop.0 |
| 0.02% | astar | astar | [.] regmngobj::addallregions |
| 0.02% | astar | astar | [.] regmngobj::definemiddlere |
| 0.02% | astar | astar | [.] regmngobj::enlargeneighbo |
| 0.02% | astar | [kernel.kallsyms] | [k] memset |
| 0.02% | astar | astar | [.] regmngobj::findfreeplace |
| 0.02% | astar | astar | [.] regobj::makebound2 |
| 0.02% | astar | [kernel.kallsyms] | [k] change_protection |
| 0.01% | astar | astar | [.] myrandom |
| 0.01% | astar | astar | [.] regboundobj::step |
| 0.01% | astar | [kernel.kallsyms] | [k] _raw_spin_unlock_irqresto |
| 0.01% | astar | [kernel.kallsyms] | [k] __handle_mm_fault |

The key function snippet:

```
void way2obj::releasepoint(i32 px, i32 py) {
  i32 x,y;
  i32 x1,y1,x2,y2;
  i32 mindist,dist;
  bool flcenter;

  if (waymap[px+py*mapsizex].fillnum==fillnum)
    return;
  // ... omitted ...
}
```

```
0000000000014876 <_ZN7way2obj12releasepointEii>:
   14876:       6785                    lui     a5,0x1
   14878:       97aa                    add     a5,a5,a0
   1487a:       1487a783                lw      a5,328(a5) # 1148 <__abi_tag-0xf14c>
   1487e:       6518                    ld      a4,8(a0)
   14880:       01055683                lhu     a3,16(a0)
   14884:       02c787bb                mulw    a5,a5,a2
   14888:       9fad                    addw    a5,a5,a1
   1488a:       078a                    sll     a5,a5,0x2
   1488c:       97ba                    add     a5,a5,a4
   1488e:       0007d703                lhu     a4,0(a5)
   14892:       30d70863                beq     a4,a3,14ba2 <_ZN7way2obj12releasepointEii+0x32c>
   14896:       7175                    add     sp,sp,-144
   14898:       f8ca                    sd      s2,112(sp)
   1489a:       e4de                    sd      s7,72(sp)
   1489c:       e0e2                    sd      s8,64(sp)
   1489e:       fc66                    sd      s9,56(sp)
   148a0:       e506                    sd      ra,136(sp)
   148a2:       f4ce                    sd      s3,104(sp)
   148a4:       ecd6                    sd      s5,88(sp)
   148a6:       f46e                    sd      s11,40(sp)
#... omitted ...
   14b5e:       640a                    ld      s0,128(sp)
   14b60:       74e6                    ld      s1,120(sp)
   14b62:       7a06                    ld      s4,96(sp)
   14b64:       6b46                    ld      s6,80(sp)
   14b66:       7d42                    ld      s10,48(sp)
   14b68:       02dc87bb                mulw    a5,s9,a3
   14b6c:       017787bb                addw    a5,a5,s7
   14b70:       963e                    add     a2,a2,a5
   14b72:       00064703                lbu     a4,0(a2)
   14b76:       078a                    sll     a5,a5,0x2
   14b78:       95be                    add     a1,a1,a5
   14b7a:       00e907b3                add     a5,s2,a4
   14b7e:       0127c783                lbu     a5,18(a5)
   14b82:       00f59123                sh      a5,2(a1)
   14b86:       6785                    lui     a5,0x1
   14b88:       993e                    add     s2,s2,a5
   14b8a:       12492783                lw      a5,292(s2)
   14b8e:       e77795e3                bne     a5,s7,149f8 <_ZN7way2obj12releasepointEii+0x182>
   14b92:       12892783                lw      a5,296(s2)
   14b96:       e79791e3                bne     a5,s9,149f8 <_ZN7way2obj12releasepointEii+0x182>
   14b9a:       4785                    li      a5,1
   14b9c:       12f90a23                sb      a5,308(s2)
   14ba0:       bda1                    j       149f8 <_ZN7way2obj12releasepointEii+0x182>
   14ba2:       8082                    ret
```

```
0000000000003fd8 <_ZN7way2obj12releasepointEii>:
    3fd8:       7155                    add     sp,sp,-208
    3fda:       e586                    sd      ra,200(sp)
    3fdc:       e1a2                    sd      s0,192(sp)
    3fde:       fd26                    sd      s1,184(sp)
    3fe0:       f94a                    sd      s2,176(sp)
    3fe2:       f54e                    sd      s3,168(sp)
    3fe4:       f152                    sd      s4,160(sp)
    3fe6:       ed56                    sd      s5,152(sp)
    3fe8:       e95a                    sd      s6,144(sp)
    3fea:       e55e                    sd      s7,136(sp)
    3fec:       e162                    sd      s8,128(sp)
    3fee:       fce6                    sd      s9,120(sp)
    3ff0:       f8ea                    sd      s10,112(sp)
    3ff2:       f4ee                    sd      s11,104(sp)
    3ff4:       872a                    mv      a4,a0
    3ff6:       6505                    lui     a0,0x1
    3ff8:       1205051b                addw    a0,a0,288 # 1120 <_ZN6wayobj11createwayarEiiRP8point16tRi+0x48>
    3ffc:       00a70833                add     a6,a4,a0
    4000:       02882503                lw      a0,40(a6)
    4004:       84ae                    mv      s1,a1
    4006:       670c                    ld      a1,8(a4)
    4008:       02c50533                mul     a0,a0,a2
    400c:       9d25                    addw    a0,a0,s1
    400e:       050a                    sll     a0,a0,0x2
    4010:       952e                    add     a0,a0,a1
    4012:       00055683                lhu     a3,0(a0)
    4016:       01075583                lhu     a1,16(a4)
    401a:       1eb68963                beq     a3,a1,420c <_ZN7way2obj12releasepointEii+0x234>
#... omitted ...
    420c:       60ae                    ld      ra,200(sp)
    420e:       640e                    ld      s0,192(sp)
    4210:       74ea                    ld      s1,184(sp)
    4212:       794a                    ld      s2,176(sp)
    4214:       79aa                    ld      s3,168(sp)
    4216:       7a0a                    ld      s4,160(sp)
    4218:       6aea                    ld      s5,152(sp)
    421a:       6b4a                    ld      s6,144(sp)
    421c:       6baa                    ld      s7,136(sp)
    421e:       6c0a                    ld      s8,128(sp)
    4220:       7ce6                    ld      s9,120(sp)
    4222:       7d46                    ld      s10,112(sp)
    4224:       7da6                    ld      s11,104(sp)
    4226:       6169                    add     sp,sp,208
    4228:       8082                    ret
    422a:       02882503                lw      a0,40(a6)
    422e:       630c                    ld      a1,0(a4)
    4230:       02c50533                mul     a0,a0,a2
    4234:       9d25                    addw    a0,a0,s1
    4236:       95aa                    add     a1,a1,a0
    4238:       0005c583                lbu     a1,0(a1)
    423c:       95ba                    add     a1,a1,a4
    423e:       0125c583                lbu     a1,18(a1)
    4242:       9d2e                    add     s10,s10,a1
    4244:       b765                    j       41ec <_ZN7way2obj12releasepointEii+0x214>
```

# Shrink Wrap Optimization

### GCC's implementation

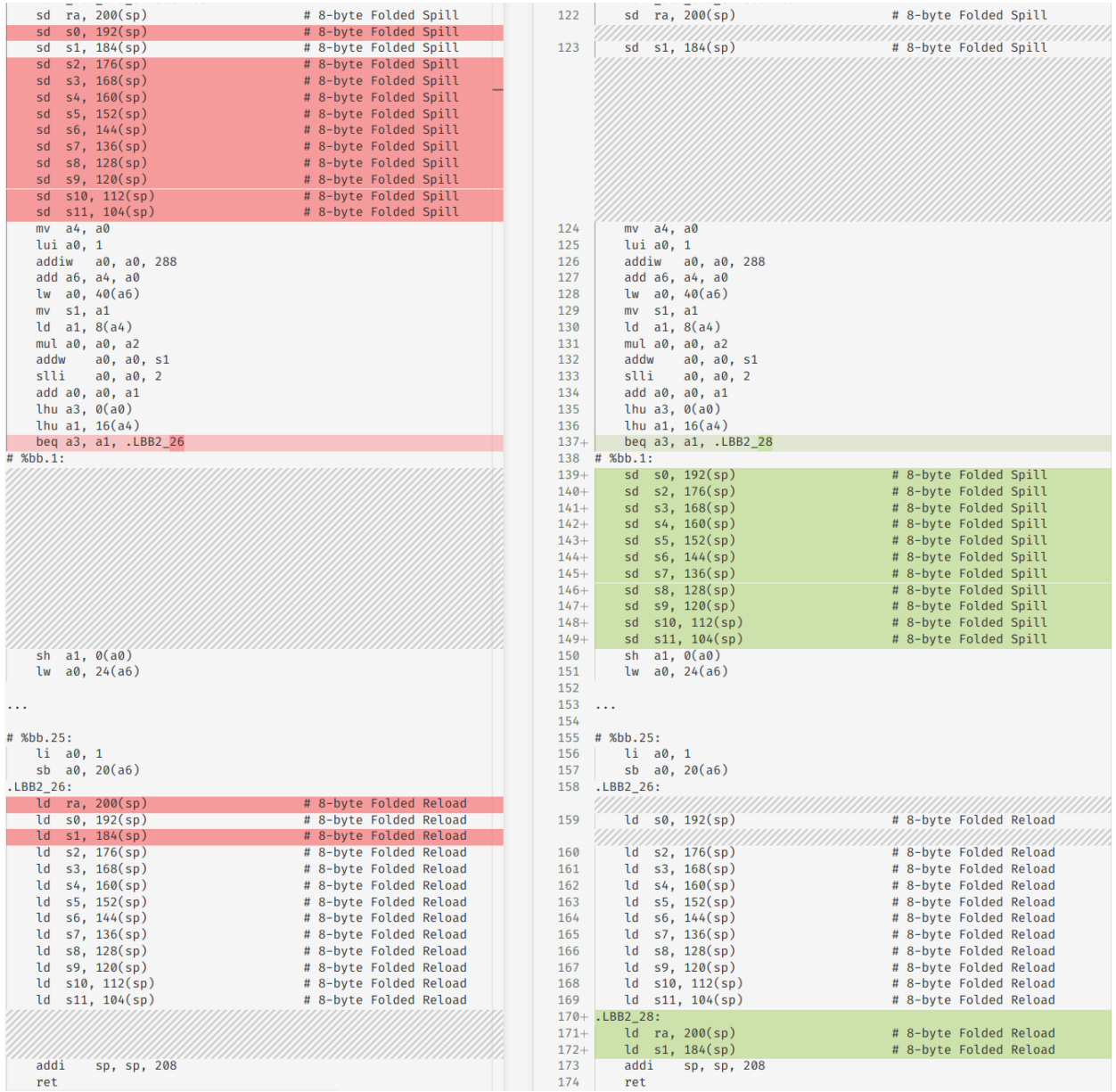
More sophisticated, resulting in fewer dynamic instructions

### LLVM's implementation

Relatively simpler, with certain optimization opportunities missed

> - Compilation time considerations
> - Potential breakage of debugging tools (like unwinding)

Manually adjusting the placement of some callee-saved register saves/restores (s2-s11) in the LLVM-generated assembly for this function, get a **3.8%** reduction in dynamic instructions for the entire 473.astar benchmark.



```
    sd  ra, 200(sp)                 # 8-byte Folded Spill
    sd  s0, 192(sp)                 # 8-byte Folded Spill
    sd  s1, 184(sp)                 # 8-byte Folded Spill
    sd  s2, 176(sp)                 # 8-byte Folded Spill
    sd  s3, 168(sp)                 # 8-byte Folded Spill
    sd  s4, 160(sp)                 # 8-byte Folded Spill
    sd  s5, 152(sp)                 # 8-byte Folded Spill
    sd  s6, 144(sp)                 # 8-byte Folded Spill
    sd  s7, 136(sp)                 # 8-byte Folded Spill
    sd  s8, 128(sp)                 # 8-byte Folded Spill
    sd  s9, 120(sp)                 # 8-byte Folded Spill
    sd  s10, 112(sp)                # 8-byte Folded Spill
    sd  s11, 104(sp)                # 8-byte Folded Spill
    mv  a4, a0
    lui a0, 1
    addiw   a0, a0, 288
    add a6, a4, a0
    lw  a0, 40(a6)
    mv  s1, a1
    ld  a1, 8(a4)
    mul a0, a0, a2
    addw    a0, a0, s1
    slli    a0, a0, 2
    add a0, a0, a1
    lhu a3, 0(a0)
    lhu a1, 16(a4)
    beq a3, a1, .LBB2_26
# %bb.1:
    sh  a1, 0(a0)
    lw  a0, 24(a6)

...

# %bb.25:
    li  a0, 1
    sb  a0, 20(a6)
.LBB2_26:
    ld  ra, 200(sp)                 # 8-byte Folded Reload
    ld  s0, 192(sp)                 # 8-byte Folded Reload
    ld  s1, 184(sp)                 # 8-byte Folded Reload
    ld  s2, 176(sp)                 # 8-byte Folded Reload
    ld  s3, 168(sp)                 # 8-byte Folded Reload
    ld  s4, 160(sp)                 # 8-byte Folded Reload
    ld  s5, 152(sp)                 # 8-byte Folded Reload
    ld  s6, 144(sp)                 # 8-byte Folded Reload
    ld  s7, 136(sp)                 # 8-byte Folded Reload
    ld  s8, 128(sp)                 # 8-byte Folded Reload
    ld  s9, 120(sp)                 # 8-byte Folded Reload
    ld  s10, 112(sp)                # 8-byte Folded Reload
    ld  s11, 104(sp)                # 8-byte Folded Reload
    addi    sp, sp, 208
    ret
```

```
122     sd  ra, 200(sp)                 # 8-byte Folded Spill
123     sd  s1, 184(sp)                 # 8-byte Folded Spill
124     mv  a4, a0
125     lui a0, 1
126     addiw   a0, a0, 288
127     add a6, a4, a0
128     lw  a0, 40(a6)
129     mv  s1, a1
130     ld  a1, 8(a4)
131     mul a0, a0, a2
132     addw    a0, a0, s1
133     slli    a0, a0, 2
134     add a0, a0, a1
135     lhu a3, 0(a0)
136     lhu a1, 16(a4)
137+    beq a3, a1, .LBB2_28
138 # %bb.1:
139+    sd  s0, 192(sp)                 # 8-byte Folded Spill
140+    sd  s2, 176(sp)                 # 8-byte Folded Spill
141+    sd  s3, 168(sp)                 # 8-byte Folded Spill
142+    sd  s4, 160(sp)                 # 8-byte Folded Spill
143+    sd  s5, 152(sp)                 # 8-byte Folded Spill
144+    sd  s6, 144(sp)                 # 8-byte Folded Spill
145+    sd  s7, 136(sp)                 # 8-byte Folded Spill
146+    sd  s8, 128(sp)                 # 8-byte Folded Spill
147+    sd  s9, 120(sp)                 # 8-byte Folded Spill
148+    sd  s10, 112(sp)                # 8-byte Folded Spill
149+    sd  s11, 104(sp)                # 8-byte Folded Spill
150     sh  a1, 0(a0)
151     lw  a0, 24(a6)
152
153 ...
154
155 # %bb.25:
156     li  a0, 1
157     sb  a0, 20(a6)
158 .LBB2_26:
159     ld  s0, 192(sp)                 # 8-byte Folded Reload
160     ld  s2, 176(sp)                 # 8-byte Folded Reload
161     ld  s3, 168(sp)                 # 8-byte Folded Reload
162     ld  s4, 160(sp)                 # 8-byte Folded Reload
163     ld  s5, 152(sp)                 # 8-byte Folded Reload
164     ld  s6, 144(sp)                 # 8-byte Folded Reload
165     ld  s7, 136(sp)                 # 8-byte Folded Reload
166     ld  s8, 128(sp)                 # 8-byte Folded Reload
167     ld  s9, 120(sp)                 # 8-byte Folded Reload
168     ld  s10, 112(sp)                # 8-byte Folded Reload
169     ld  s11, 104(sp)                # 8-byte Folded Reload
170+ .LBB2_28:
171+    ld  ra, 200(sp)                 # 8-byte Folded Reload
172+    ld  s1, 184(sp)                 # 8-byte Folded Reload
173     addi    sp, sp, 208
174     ret
```

In summary, we now recognize that Shrink Wrap optimization plays a significant role in 473.astar.

Notably, active optimization efforts are underway in LLVM:

- #119359 [llvm] Support save/restore point splitting in shrink-wrap by enoskova-sc
- #90819 [RISCV][WIP] Let RA do the CSR saves by mgudim

# 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.

- Enhanced Automation Pipeline
- Intelligent Data Presentation
- LLVM Version Benchmarking & Regression Guard

# 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

# Thanks