开展编译器性能分析和优化需要做哪些准备

李永泰
中国科学院软件研究所 PLCT 实验室
liyongtai@iscas.ac.cn

2025/3/22

目录

- **编译环境的搭建**
- SPEC CPU **测试套件的编译**
- **测试运行**
- perf **和** objdump **工具介绍**
- **两个例子**

一些背景

- RISC-V
- Linux
- GCC
- LLVM
- SPEC CPU 2017/2006
- Code size / Dynamic instruction count

编译环境的搭建

- 硬件
- 通过包管理安装 gcc/clang 等工具
- 编译 GNU 工具链
- 编译 LLVM 工具链

```
debian@revyos-pioneer:~$ uname -a
Linux revyos-pioneer 6.1.72-pioneer #2024.01.19.06.06+4005122ed SMP Fri Jan 19 07:04:20 UTC 2024 riscv64 GNU/Linux
debian@revyos-pioneer:~$ fastfetch
       _,met$$$$$gg.          debian@revyos-pioneer
    ,g$$$$$$$$$$$$$$$P.       ---------------------
  ,g$$P"        """Y$$.".     OS: Debian GNU/Linux bookworm 12.0 riscv64
 ,$$P'              `$$$.     Host: Sophgo Mango
',$$P       ,ggs.     `$$b:   Kernel: 6.1.72-pioneer
`d$$'     ,$P"'   .    $$$    Uptime: 7 days, 10 hours, 33 mins
 $$P      d$'     ,    $$$P   Packages: 1333 (dpkg)
 $$:      $.   -    ,d$$'     Shell: bash 5.2.15
 $$;      Y$b._   _,d$P'      Cursor: Adwaita
 Y$$.    `.`"Y$$$$P"'         Terminal: /dev/pts/0
 `$$b      "-.__              CPU: rv64gc (64)
  `Y$$                        GPU: AMD Radeon HD 6450/7450/8450 / R5 230 OEM
   `Y$$.                      Memory: 715.00 MiB / 124.88 GiB (1%)
     `$$b.                    Swap: Disabled
       `Y$$b.                 Disk (/): 370.38 GiB / 937.56 GiB (40%) - ext4
          `"Y$b._             Local IP (enP2p129s0f1): 192.168.16.119/24 *
              `"""            Locale: en_GB.UTF-8

debian@revyos-pioneer:~$
```

```
debian@revyos-pioneer:~$ gcc --version
gcc (Debian 13.2.0-4revyos1) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

debian@revyos-pioneer:~$ clang --version
Debian clang version 14.0.6
Target: riscv64-unknown-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
debian@revyos-pioneer:~$ cmake --version
cmake version 3.25.1

CMake suite maintained and supported by Kitware (kitware.com/cmake).
debian@revyos-pioneer:~$ ninja --version
1.11.1
debian@revyos-pioneer:~$
```

构建 GCC

依赖的基本工具：gcc, g++, make, binutils, libtool, bzip2, gzip, tar, perl
依赖的一些库：GMP、MPFR、MPC

- 使用系统包管理工具安装：libgmp-dev libmpfr-dev libmpc-dev (debian)
- gcc 源码中提供的下载工具

*https://gcc.gnu.org/wiki/InstallingGCC
*https://gcc.gnu.org/install/prerequisites.html

构建 GCC

获取源码

下载源码包

https://gcc.gnu.org/mirrors.html

https://ftp.gnu.org/gnu/gcc/gcc-14.2.0/

或者从 git 获取

```
git clone https://github.com/gcc-mirror/gcc
# git://gcc.gnu.org/git/gcc.git
# checkout 到感兴趣的 commit/branch
```

下载依赖库

```
./contrib/download_prerequisites
```

构建 GCC

- 创建独立构建目录
  防止污染源码，编译不同版本时需要清理/创建多个 build 目录

```
mkdir ../build && cd ../build
```

- 运行 configure

```
../gcc/configure --prefix=/path/to/install -enable-languages=c,c++,fortran
```

> 有非常多的配置选项，可以使用 configure --help 查看

- 编译安装

```
make -j$(nproc)
make install
```

构建 LLVM

- 构建 LLVM 的依赖相对较少，准备好 c/c++ 编译器,cmake,ninja 即可
- 获取源码

```
git clone https://github.com/llvm/llvm-project.git
# https://github.com/llvm/llvm-project/releases/
```

构建 LLVM

- 创建独立构建目录并运行 cmake

```
mkdir build && cd build

cmake -DLLVM_TARGETS_TO_BUILD="RISCV" -DLLVM_ENABLE_PROJECTS="clang;flang;mlir"  \
-DCMAKE_BUILD_TYPE=Release -DCMAKE_C_COMPILER=clang  -DCMAKE_CXX_COMPILER=clang++ \
-DCMAKE_INSTALL_PREFIX=/path/to/install -G Ninja \
-DLLVM_PARALLEL_LINK_JOBS=N -DLLVM_USE_LINKER=mold ../llvm
```

- 编译安装

```
ninja -j$(nproc)
ninja install
```

> 编译 LLVM 链接时所需内存较大，尤其是 Debug 版本，需要控制并行链接数

*https://llvm.org/docs/CMake.html

SPEC CPU 概览

- 测试项目分成两类:
    - **INT (Integer Performance)**: 编译、压缩，逻辑运算.
    - **FP (Floating-Point Performance)**: 科学计算, 图像处理.
- 两个模式:
    - **SPECspeed**: Single-threaded optimization.
    - **SPECrate**: Multi-core/thread performance.

Int 类型测试程序

SPECrate2017 Integer	SPECspeed2017 Integer	Language	KLOC	Application Area
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1,304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

fp 类型测试程序

SPECrate2017 Floating Point	SPECspeed2017 Floating Point	Language	KLOC	Application Area
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity
508.namd_r		C++	8	Molecular dynamics
510.parest_r		C++	427	Biomedical imaging: optical tomography with finite elements
511.povray_r		C++, C	170	Ray tracing
519.lbm_r	619.lbm_s	C	1	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	991	Weather forecasting
526.blender_r		C++, C	1,577	3D rendering and animation
527.cam4_r	627.cam4_s	Fortran, C	407	Atmosphere modeling
	628.pop2_s	Fortran, C	338	Wide-scale ocean modeling (climate level)
538.imagick_r	638.imagick_s	C	259	Image manipulation
544.nab_r	644.nab_s	C	24	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	Computational Electromagnetics
554.roms_r	654.roms_s	Fortran	210	Regional ocean modeling

两种测试模式

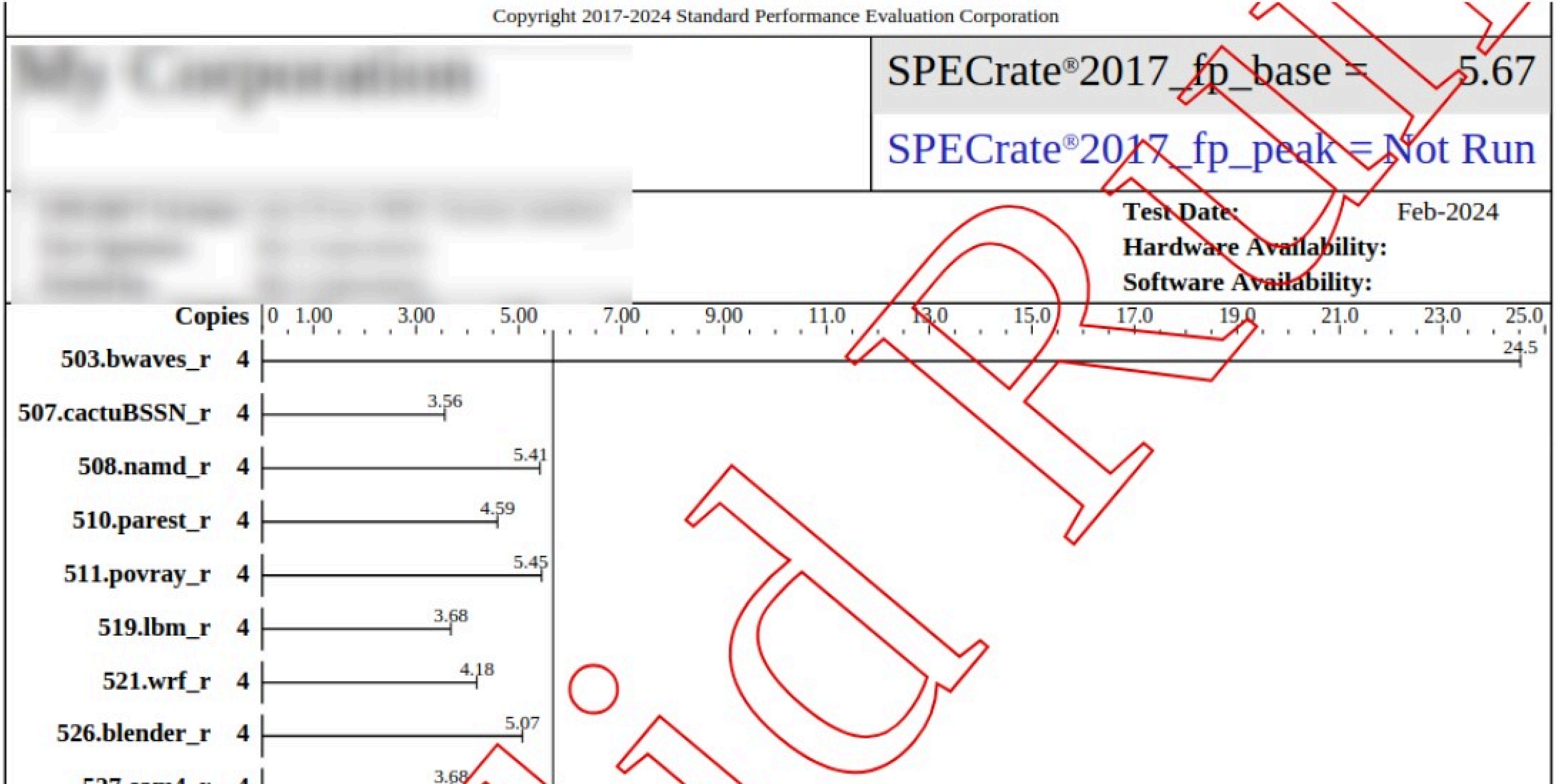
SPECspeed Metrics

- 1 copy of each benchmark in a suite is run.
- The tester may choose how many OpenMP threads to use.
- For each benchmark, a performance ratio is calculated as:
  `time on a reference machine / time on the SUT`
- Higher scores mean that less time is needed.
- **Example:**
  - The reference machine ran 600.perlbench_s in 1775 seconds.
  - A particular SUT took about 1/5 the time, scoring about 5.
  - More precisely: 1775/354.329738 = 5.009458

SPECrate Metrics

- The tester chooses how many concurrent copies to run
- OpenMP is disabled.
- For each benchmark, a performance ratio is calculated as:
  `number of copies * (time on a reference machine / time on the SUT)`
- Higher scores mean that more work is done per unit of time.
- **Example:**
  - The reference machine ran 1 copy of 500.perlbench_r in 1592 seconds.
  - A particular SUT ran 8 copies in about 1/3 the time, scoring about 24.
  - More precisely: 8*(1592/541.52471) = 23.518776

跑分结果



Copyright 2017-2024 Standard Performance Evaluation Corporation
SPECrate®2017_fp_base = 5.67
SPECrate®2017_fp_peak = Not Run
Test Date: Feb-2024
Hardware Availability:
Software Availability:
Copies
0 1.00 3.00 5.00 7.00 9.00 11.0 13.0 15.0 17.0 19.0 21.0 23.0 25.0
503.bwaves_r 4 24.5
507.cactuBSSN_r 4 3.56
508.namd_r 4 5.41
510.parest_r 4 4.59
511.povray_r 4 5.45
519.lbm_r 4 3.68
521.wrf_r 4 4.18
526.blender_r 4 5.07
3.68

SPEC CPU 的安装

- 获取安装包，通常情况下是一个 *.iso 镜像文件
- 提取文件
- 打 patch 编译工具集（可选）
- 安装、验证

哪些情况需要自己编译 RISC-V 工具集

1. SPEC CPU 2006
2. SPEC CPU 2017, version < 1.0.9

> 低版本可以通过 `runcpu -update` 在线升级到最新版本
> 但前提是当前有可用的 `runcpu` 工具

*https://www.spec.org/cpu2017/Docs/tools-build.html

提取安装文件和 tools 源码

```
mount /path/to/cpu2017.iso /mnt -o loop
mkdir /home/test/spec2017
cp -r /mnt/* /home/test/spec2017_iso

chmod -R +w /home/test/spec2017_iso #增加写权限

export SPEC=/home/test/spec2017_iso
cd $SPEC/install_archives
tar xf tools-src.tar -C .. #解压工具集源码
```

给源码打补丁

1. config.guess 和 config.sub 太过老旧，不能识别 riscv 架构

下载最新版本的 config.guess 和 config.sub

http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.guess
http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.sub

替换以下位置

```
$SPEC/tools/src/specinvoke/
$SPEC/tools/src/specsum/build-aux/
$SPEC/tools/src/tar-1.28/build-aux/
$SPEC/tools/src/make-4.2.1/config/
$SPEC/tools/src/rxp-1.5.0/
$SPEC/tools/src/expat-2.1.0/conftools/
$SPEC/tools/src/xz-5.2.2/build-aux/
```

给源码打补丁

2. 编译 perl 时，会将 gcc 10 识别为 gcc 1.x

这是因为 $SPEC/tools/src/perl-5.24.0/Configure 和 $SPEC/tools/src/perl-5.24.0/cflags.SH 中使用 1* 匹配 gcc 版本号，需改为 1.*

```
 fi
 $rm -f try try.*
 case "$gccversion" in
-1*) cpp=`./loc gcc-cpp $cpp $pth` ;;
+1.*) cpp=`./loc gcc-cpp $cpp $pth` ;;
 esac
 case "$gccversion" in
 '') gccosandvers='' ;;
@@ -5438,7 +5438,7 @@ fi
 case "$hint" in
 default|recommended)
        case "$gccversion" in
-       1*) dflt="$dflt -fpcc-struct-return" ;;
+       1.*) dflt="$dflt -fpcc-struct-return" ;;
        esac
        case "$optimize:$DEBUGGING" in
        *-g*:old) dflt="$dflt -DDEBUGGING";;
@@ -5453,7 +5453,7 @@ default|recommended)
```

给源码打补丁

3. $SPEC/tools/src/TimeDate-2.30/t/getdate.t 测试失败 需要做以下修改

```
--- getdate.t
+++ getdate.t
@@ -156,7 +156,7 @@ Jul 22 10:00:00 UTC 2002         ;1027332000
 !;

 require Time::Local;
-my $offset = Time::Local::timegm(0,0,0,1,0,70);
+my $offset = Time::Local::timegm(0,0,0,1,0,1970);

 @data = split(/\n/, $data);
```

*https://github.com/sihuan/llvm-work/blob/master/spec2017/README.md

编译并打包工具集

1. 编译, 使用 $SPEC/tools/src/buildtools

```
export MAKEFLAGS=-j4
./buildtools
```

2. 打包，使用 $SPEC/bin/packagetools

```
mkdir -p $SPEC/tools/bin/linux-riscv64
echo '……' > $SPEC/tools/bin/linux-riscv64/description
./bin/packagetools linux-riscv64
```

安装 SPEC CPU

```
./install.sh -u linux-riscv64 -d /home/test/spec2017
```

测试

```
source shrc
runcpu --version
SPEC CPU(r) 2017 Benchmark Suites
Copyright 1995-2019 Standard Performance Evaluation Corporation (SPEC)

runcpu v6612
Using 'linux-riscv64' tools
                 This is the SPEC CPU2017 benchmark tools suite.
```

编写成对的配置文件

1. base 和 peak 两种配置
2. 主要需要修改的地方
    - 编译器路径
    - -march
    - 不同的优化选项、优化等级

- 可以查看 Example-* 开头的样例配置

```
$ diff llvm-c9a6e993f7b3-rv64gc_zba_zbb_zbs.cfg gcc-d28ea8e5a704-rv64gc_zba_zbb_zbs.cfg
1c1
< %define commit c9a6e993f7b3
---
> %define commit d28ea8e5a704
<-SKIP->
60,65c60,65
<    preENV_LD_LIBRARY_PATH  = %{llvm_dir}/lib/:/lib
<    SPECLANG                = %{llvm_dir}/bin/
<    CC                      = $(SPECLANG)clang -std=c99 -Wno-implicit-int
<    CXX                     = $(SPECLANG)clang++ -std=c++03
<    FC                      = $(SPECLANG)flang-new
---
>
>    preENV_LD_LIBRARY_PATH  = %{gcc_dir}/lib/:/lib
>    SPECLANG                = %{gcc_dir}/bin/
>    CC                      = $(SPECLANG)gcc   -L%{gcc_dir}/lib  -std=c99 -Wno-implicit-int
>    CXX                     = $(SPECLANG)g++   -L%{gcc_dir}/lib  -std=c++03
>    FC                      = $(SPECLANG)gfortran -L%{gcc_dir}/lib
```

跑个分看看

```
source shrc
ulimit -s unlimited
runcpu --config=xxx --size=ref --tuning=all all
```

测试范围可以形如

- 503
- 519.lbm_r
- int
- fprate

*如果想要生成一个合法的报告，需要跑多次（默认），test size 需要是 ref，tuning 可以是 base 或者 all

*https://www.spec.org/cpu2017/Docs/runcpu.html

分步运行

- build —— 仅**编译**指定的测试项目
- runsetup —— 构建测试运行环境

仅编译测试

使用以下命令编译测试并打包编译得到的可执行程序

```
runcpu --config label.cfg --action build intspeed
spectar cvf - config/label.cfg benchspec/CPU/*/exe/*label | specxz -T 0 > label.tar.xz
```

> 上边的 ***label*** 形如 `llvm-c9a6e993-rv64gc_zba_zbb_zbs` or `gcc-d28ea8e5-rv64gcv_zba_zbb_zbs`.

> action 选项:
>
> - build: 构建指定套件
> - buildsetup: 生成构建目录，复制源码，生成 Makefile

```
benchspec                                                          17.3 MiB
  CPU                                                              17.3 MiB
    500.perlbench_r                                                2.2 MiB
      exe                                                          2.2 MiB
        perlbench_r_base.llvm-c9a6e993f7b3-rv64gc_zba_zbb_zbs-64   2.2 MiB
    502.gcc_r                                                      9.9 MiB
      exe                                                          9.9 MiB
        cpugcc_r_base.llvm-c9a6e993f7b3-rv64gc_zba_zbb_zbs-64      9.9 MiB
    503.bwaves_r                                                   1.8 MiB
    SKIP                                                           3.4 MiB
config                                                             60.9 KiB
  llvm-c9a6e993f7b3-rv64gc_zba_zbb_zbs.cfg                         60.9 KiB
```

生成运行环境

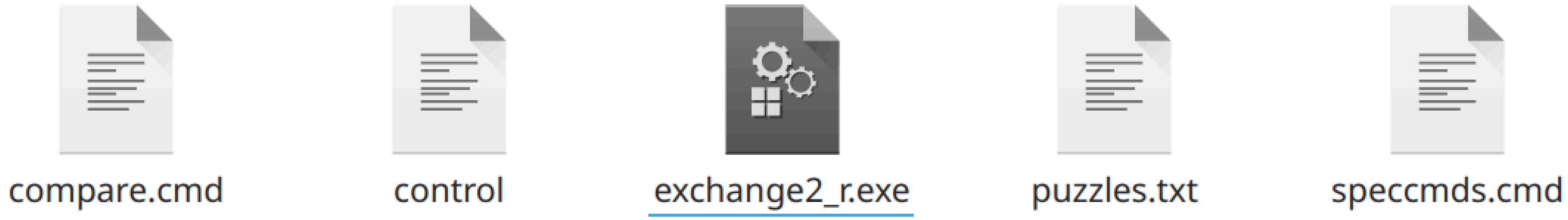
生成运行环境，包括测试需要的输入和 `speccmds.cmd` 等文件.

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

> `--action runsetup`选项生成运行目录，复制二进制文件和数据文件，创建控制文件

> 测试数据和控制文件是架构无关的，但是 test size 有关（ref/test）

548_exchage2_r 的运行目录



- `exchage2_r.exe`: 待测试的可执行程序
- `puzzles.txt`: 输入数据
- `control`, `speccmds.cmd`, `compare.cmd`: 一些控制文件，描述了测试如何运行，如何对比数据

生成运行环境

使用 `specinvoke` 可以查看 `speccmds.cmd` 中关于测试如何运行的描述

```
$   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
```

现在我们已经可以手动去运行任意一个测试。

为什么选用 QEMU

1. RISC-V 机器目前还是比较慢
2. 很多测试项有多个 workload，可以设计成并行运行
3. 借助 QEMU 的插件系统可以方便的统计指令数（也可以自己编写插件统计更详细的信息）

运行测试并获取数据

- Code Size: strip 可执行文件然后直接统计文件大小
- 动态指令数 —— 使用 QEMU User 运行并统计
    - 构建 QEMU 和 insn 插件
    - 带 insn 插件运行测试

构建带有 TCG 插件的 QEMU

获取源码

```
git clone https://github.com/qemu/qemu.git && cd qemu
mkdir build && cd build

../configure --prefix=/path/to/install \
   --target-list=riscv64-linux-user --enable-plugins

make
make install
```

*https://www.qemu.org/docs/master/devel/index-build.html

*https://www.qemu.org/docs/master/devel/tcg-plugins.html

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

使用带有 insn 插件的 QEMU

```
$ path/to/qemu-riscv64 -plugin path/to/plugin/libinsn.so -d plugin ./demo
  cpu 0 insns: 20250322
  total insns: 20250322
```

更多的选项

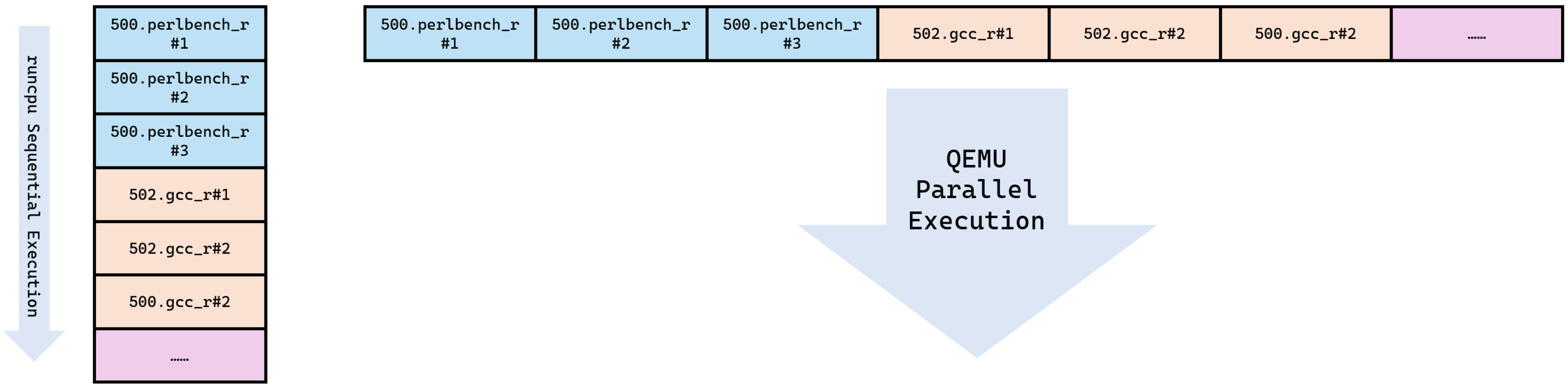
实际运行 SPEC CPU 测试的话，需要更多的选项，比如

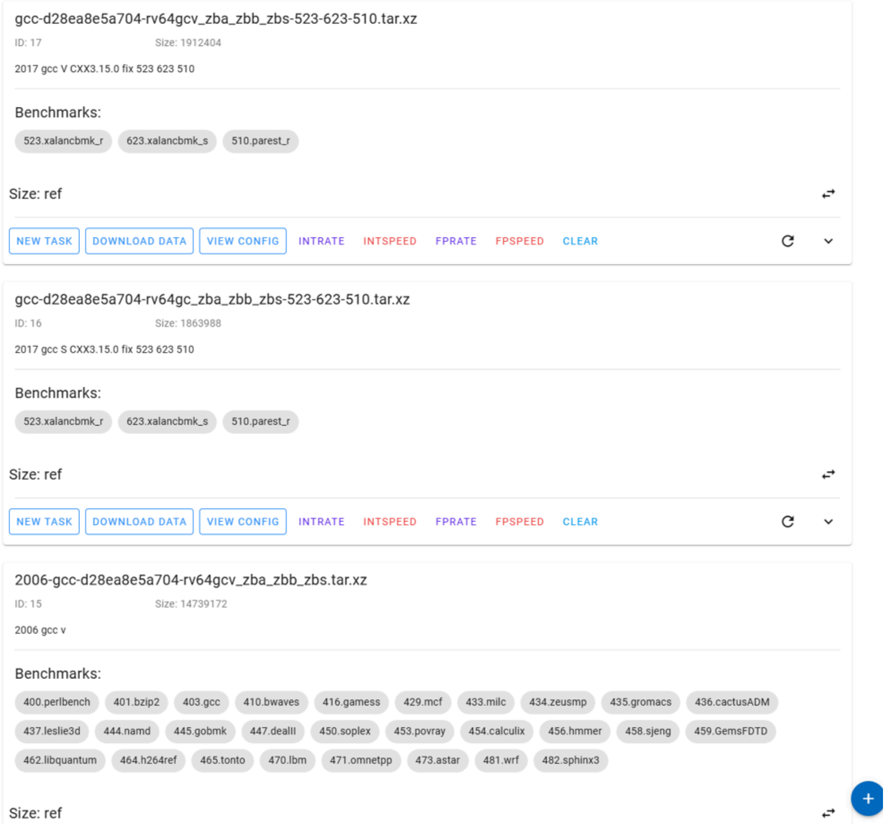
- -L /opt/riscv/sysroot
- -cpu rv64,v=true,vext_spec=v1.0
- -s 819200000000

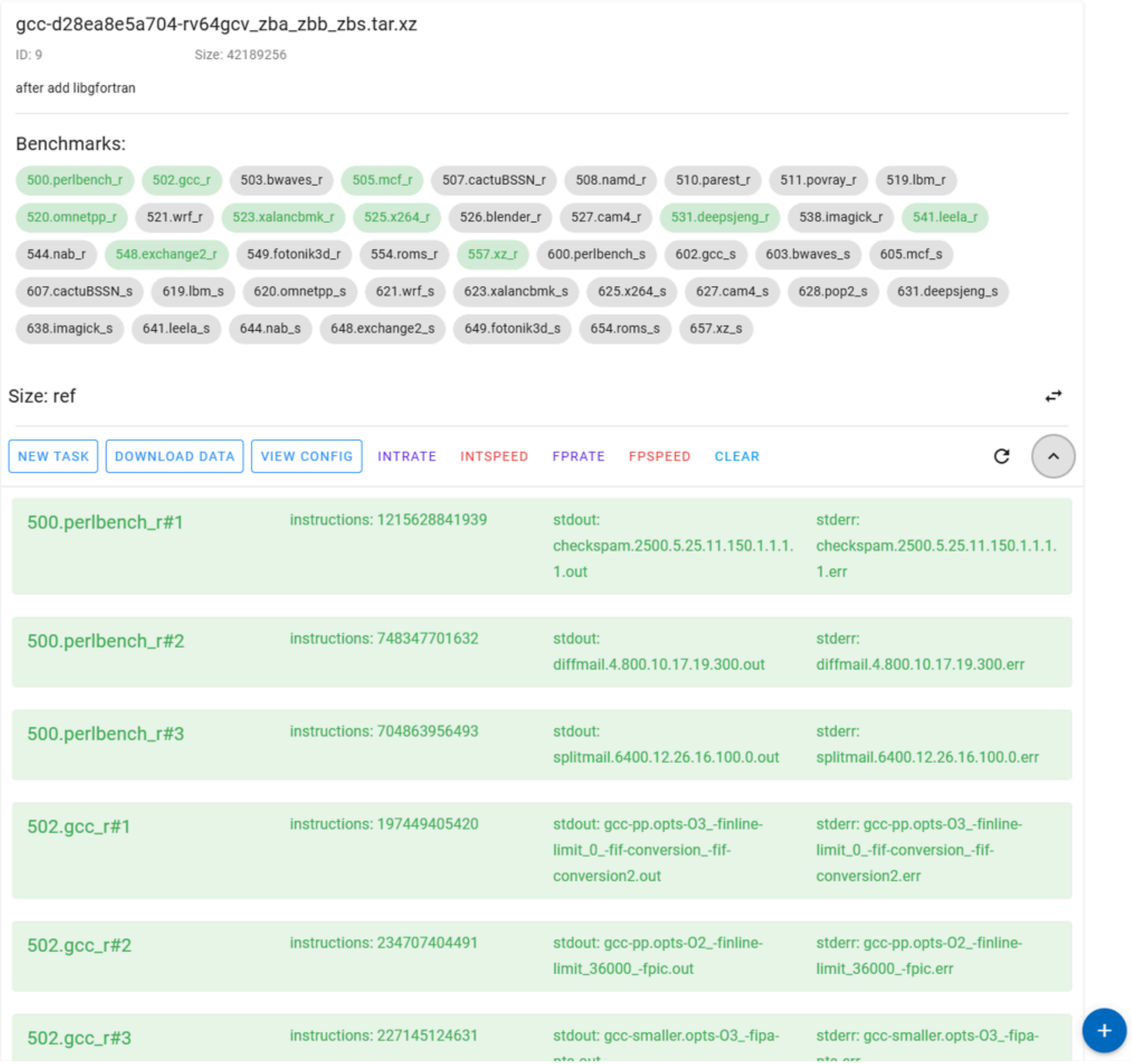
不过可以用自动化工具来代劳

开源的自动化工具

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.out
stderr: checkspam.2500.5.25.11.150.1.1.1.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-

统计得到的数据

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

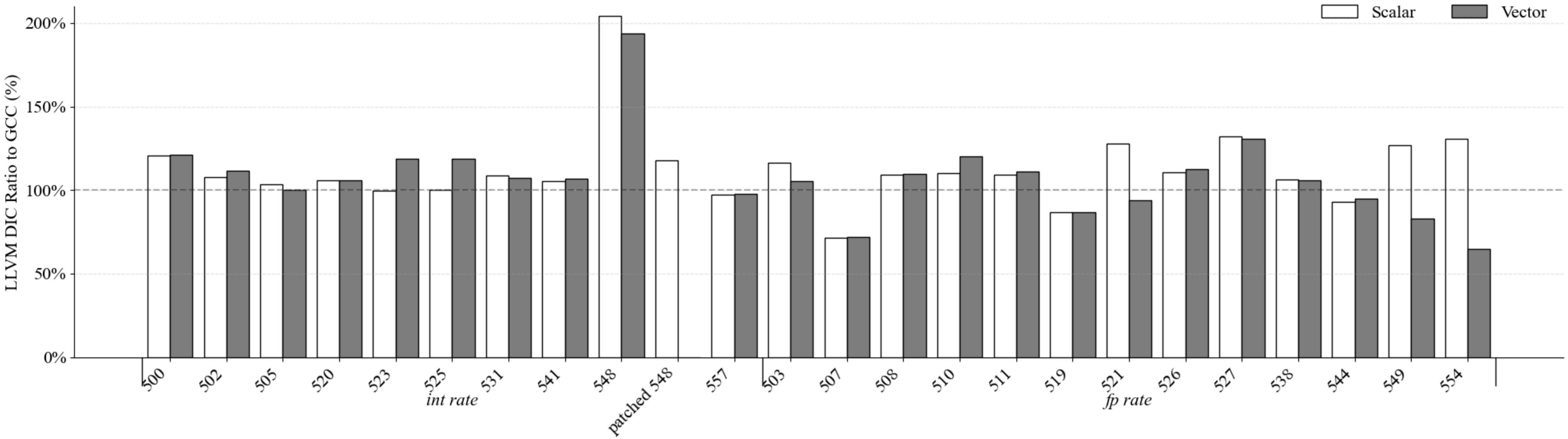
Range	C/C++	Fortran
< 0.9	508,541,507,519,531 557,538,525,505,544	527
0.9-1	500,502	
1-1.1	511,510,520	
> 1.1	523,526	521,554,549,548,503

Google表格

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%

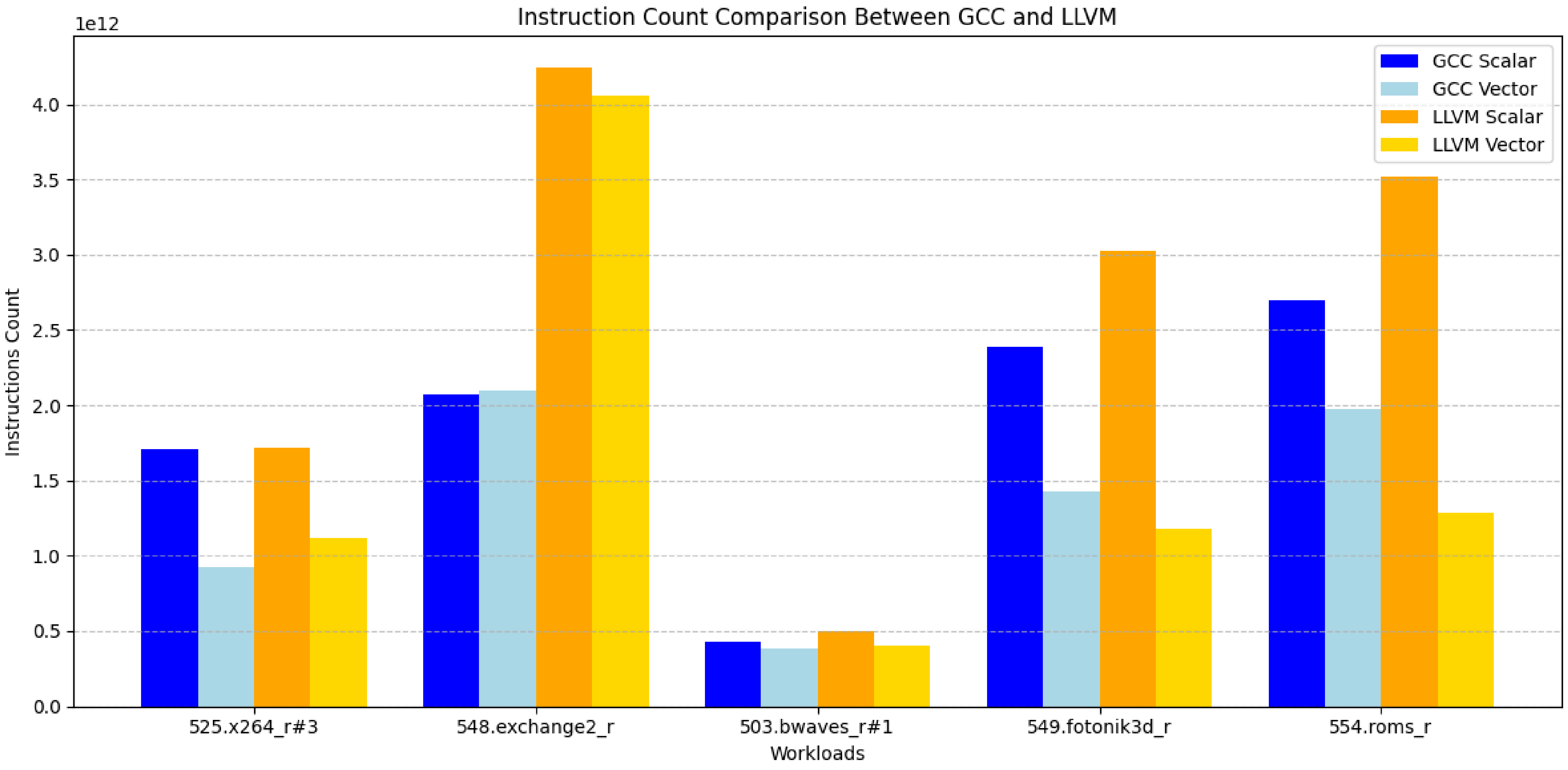
Benchmark	gcc s	gcc v	llvm s	llvm v	llvms/gccs	llvmv/gccv
527.cam4_r	18509560	19053760	11304944	11878504	61.08%	62.34%
521.wrf_r	30947416	35897608	41985072	48908776	135.67%	136.25%
554.roms_r	840072	1311384	2109832	2393064	251.15%	182.48%
549.fotonik3d_r	299200	326704	891232	957944	297.87%	293.21%
548.exchange2_r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

动态指令数



Scalar
Vector
LLVM DIC Ratio to GCC (%)
200%
150%
100%
50%
0%
500
502
505
520
523
525
531
541
548
patched 548
557
503
507
508
510
511
519
521
526
527
538
544
549
554
int rate
fp rate

动态指令数



Instruction Count Comparison Between GCC and LLVM
1e12
Instructions Count
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
GCC Scalar
GCC Vector
LLVM Scalar
LLVM Vector
525.x264_r#3
548.exchange2_r
503.bwaves_r#1
549.fotonik3d_r
554.roms_r
Workloads

动态指令数

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%

性能分析工具

- perf —— 动态分析，指令数、分支数、热点函数
- objdump —— 静态分析，反汇编，局部分析

perf

- Linux 内核自带的性能分析工具
- 基于硬件性能计数器、跟踪点（tracepoints）进行采样和分析

核心功能:

- CPU 性能分析（指令、缓存、分支预测）
- 函数级代码热点分析
- 系统级资源监控

安装和配置

不同发行版打包情况不同

```
apt install linux-perf # revyos

pacman -S perf # archlinux
```

配置权限

```
# 允许非 root 用户使用
sudo sysctl -w kernel.perf_event_paranoid=1
```

常用命令

- perf list 列出支持的事件
- perf stat 统计程序运行时的事件
- perf record 采样并保存数据（生成 perf.data）
- perf report 分析采样数据
- perf top 实时显示系统热点

其他选项

- perf stat -e 选项可以指定关心的事件
- perf record -F 选项可以控制采样频率

objdump

GNU Binutils 工具集的一部分，用于解析二进制文件（可执行文件、目标文件、共享库）。
支持反汇编、查看符号、调试信息等等

和 perf 是一个互补的关系，通过 perf 定位运行时的性能问题，然后通过 objdump 反汇编分析代码静态结构，验证编译器优化效果。

安装和基础用法

安装

```
apt install binutils #一般已经预装/ debian
pacman -S riscv64-linux-gnu-binutils #x86 Archlinux
```

基础用法

```
objdump [选项] <二进制文件>
```

常见选项

```
-d, --disassemble   反汇编代码段
-S, --source        同时显示源码与汇编
-t, --syms          查看符号表
-j, --section       分析指定的节（如 .text, .data）
```

demo 程序分析

```
static inline void bar() {
        return;
}

int main() {
        int i;
        for(i = 0; i < 100000000; i++)
                bar();
        return 0;
}
```

1. 指令数 16xxxxxxxxx 是怎么来的
2. 反汇编指定函数 bar
3. 通过符号表检查内联效果

实际案例分析 548/473

548.exchange2_r 是一个针对 9*9 数独的解题程序，使用 Fortran95 编写，约有 1600 行代码，代码严重依赖递归（最高达 8 层），另外该程序不包括任何浮点运算。

无论是否开启 v 拓展 LLVM 和 GCC 的差异都巨大。

实际上，这个问题和 B 拓展也没关系，甚至和 RISC-V 架构无关

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

首先要解决的一些问题

- 使用 perf 分析的话需要在物理机器上，但是 SG2042 并没有 B 拓展
  - 尝试在 rv64gc 上复现.
- 手动编译流程(--action buildsetup / 查看 make.out)

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

- 手动测试流程(specinvoke 读取)

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

用 `perf` 在 test size 下录得的数据:

```
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
```

反汇编分析

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
```

推测原因

在 GCC 上, 热点函数 `digits_2` 被拆成了几个特化版本. 这种拆分特化是过程间常量传播优化造成的 (IPA-CP)。 此优化的主要作用之一就是条件分支消除。

也正是这样，特化的函数静态汇编行数也较少。

LLVM 里对应的优化是 `IPSCCP` Pass.

赞美 LLM

验证推测

通过使用 `-fno-ipa-cp` 选项关闭 gcc 优化

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

尝试在 LLVM 里重复优化

```
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

指令数减少且通过反汇编可以看到 `digits_2` 被特化成形如 `_QMbrute_forcePdigits_2.specialized.3` 的几个函数。

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

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

```cpp
PassBuilder::buildO1FunctionSimplificationPipeline(OptimizationLevel Level,
  // tests in ICP sequences.
  if (Phase == ThinOrFullLTOPhase::ThinLTOPostLink)
    MPM.addPass(LowerTypeTestsPass(nullptr, nullptr, true));

  invokePipelineEarlySimplificationEPCallbacks(MPM, Level);

  // Interprocedural constant propagation now that basic cleanup has occurred
  // and prior to optimizing globals.
  // FIXME: This position in the pipeline hasn't been carefully considered in
  // years, it should be re-analyzed.
  MPM.addPass(IPSCCPPass(
              IPSCCPOptions(/*AllowFuncSpec=*/
                            Level != OptimizationLevel::Os &&
                            Level != OptimizationLevel::Oz &&
                            !isLTOPreLink(Phase))));

  // Attach metadata to indirect call sites indicating the set of functions
  // they may target at run-time. This should follow IPSCCP.
  MPM.addPass(CalledValuePropagationPass());

```

6 llvm/lib/Passes/PassBuilderPipelines.cpp

```diff
@@ -1204,6 +1204,12 @@ PassBuilder::buildModuleSimplificationPipeline(OptimizationLevel Level,
   else
     MPM.addPass(buildInlinerPipeline(Level, Phase));
 
+  MPM.addPass(IPSCCPPass(
+              IPSCCPOptions(/*AllowFuncSpec=*/
+                            Level != OptimizationLevel::Os &&
+                            Level != OptimizationLevel::Oz &&
+                            !isLTOPreLink(Phase))));
+
   // Remove any dead arguments exposed by cleanups, constant folding globals,
   // and argument promotion.
   MPM.addPass(DeadArgumentEliminationPass());
```

`IPSCCP` Pass 默认在较早的位置启用，尝试重复运行 Pass 可以得到一个比较好的效果。

```
$ 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>:
```

workaround 效果：和 GCC 表型类似，性能提升 ~40%

重新用 `perf` 在 rv64gc 上统计的 `exchange2_r 0` 数据

额外的 x86_64 数据

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

473 分析

1. perf stat 查看动态指令数对比
2. perf record/report 定位问题函数
3. objdump 局部静态分析
4. 通过汇编上的差异 推断 关联的优化技术（可以请 DeepSeek 帮忙）
5. 阅读源码，编写较小的复现程序

总结和拓展

1. 构建待测的编译器
2. 构建测试套件
3. 运行并统计指标信息
4. 对比数据，使用 perf 动态分析，缩小范围
5. 反汇编，分析代码，找到汇编层面可能造成性能差异的点
6. 反向查找对应的优化技术
7. 构建复现样例

Code Size 数据: https://docs.google.com/spreadsheets/d/1e6sAkT1kZa8LQo4MWgT-NomF8fSHnClrJMVTrxktUAM
动态指令数数据:
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

Thanks