Compilers Laboratory

DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

ICEx Instituto de Ciências Exatas

UFMG

SPEC CPU 2017: Use cases

Andrei Rimsa Álvares
andrei@cefetmg.br

Summary

- Introduction
- Installing
- Configuring
- Validating
- Profiling
  - Dynamic profiling
  - Feedback-directed optimizations (FDO)

Compilers Laboratory

DEPARTMENT OF COMPUTER SCIENCE
FEDERAL UNIVERSITY OF MINAS GERAIS, BRAZIL

Introduction

Introdution

- SPEC♤ (*Standard Performance Evaluation Corporation*) is a non-profit organization responsible for benchmarks that evaluates performance and energy efficiency of computer systems
  - Cloud
  - CPU
  - Graphics/Workstations
  - High Performance Computing
  - Java Client/Server
  - Machine Learning
  - Storage
  - Power
  - Virtualization

We are interested in SPEC CPU.

♤: https://www.spec.org/

SPEC CPU

- SPEC CPU has a long history
  - SPEC CPU 92
  - SPEC CPU 95
  - SPEC CPU 2000
  - SPEC CPU 2006
  - SPEC CPU 2017♧

We are interested in the latest release: SPEC CPU 2017.

Info: a license is required to use SPEC CPU 2017.

♧: https://www.spec.org/cpu2017/

SPEC CPU 2017

- SPEC CPU 2017 contains a set of CPU intensive suites for measuring and comparing compute intensive performance, stressing a system's processor, memory subsystem and compiler

- It is package with 43 benchmarks organized into four suites
  - SPECspeed® 2017 Integer
  - SPECspeed® 2017 Floating Point
  - SPECrate® 2017 Integer
  - SPECrate® 2017 Floating Point

Compare time for a computer to complete single tasks.

Measure the throughput or work per unit of time.

spec®

SPEC Suites

- Overview of SPEC suites◊

base: must use the same compiler flags.
peak: may use different compiler flags.

Short Tag	Suite	Contents	Metrics	How many copies? What do Higher Scores Mean?
intspeed	SPECspeed® 2017 Integer	10 integer benchmarks	SPECspeed®2017_int_base SPECspeed®2017_int_peak SPECspeed®2017_int_energy_base SPECspeed®2017_int_energy_peak	SPECspeed suites always run one copy of each benchmark. Higher scores indicate that less time is needed.
fpspeed	SPECspeed®2017 Floating Point	10 floating point benchmarks	SPECspeed®2017_fp_base SPECspeed®2017_fp_peak SPECspeed®2017_fp_energy_base SPECspeed®2017_fp_energy_peak	
intrate	SPECrate® 2017 Integer	10 integer benchmarks	SPECrate®2017_int_base SPECrate®2017_int_peak SPECrate®2017_int_energy_base SPECrate®2017_int_energy_peak	SPECrate suites run multiple concurrent copies of each benchmark. The tester selects how many. Higher scores indicate more *throughput* (work per unit of time).
fprate	SPECrate® 2017 Floating Point	13 floating point benchmarks	SPECrate®2017_fp_base SPECrate®2017_fp_peak SPECrate®2017_fp_energy_base SPECrate®2017_fp_energy_peak	

The "Short Tag" is the canonical abbreviation for use with `runcpu`, where context is defined by the tools. In a published document, context may not be clear.
To avoid ambiguity in published documents, the Suite Name or the Metrics should be spelled as shown above.

We will reference some collection of benchmarks by their short name.

◊: https://www.spec.org/cpu2017/Docs/overview.html#suites

SPECinteger

- Collection of SPEC Integer benchmarks

SPECrate®2017 Integer	SPECspeed®2017 Integer	Language [1]	KLOC [2]	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

SPECrate

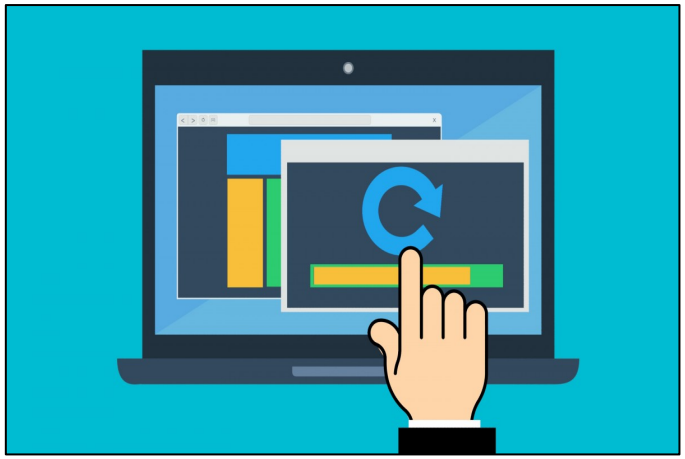
- Collection of SPEC Floating Point benchmarks

SPECrate®2017 Floating Point	SPECspeed®2017 Floating Point	Language [1]	KLOC [2]	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

[1] For multi-language benchmarks, the first one listed determines library and link options (details)

[2] KLOC = line count (including comments/whitespace) for source files used in a build / 1000

Installing



Prerequisites

- Ensure we have compilers for C, C++ and Fortran
  - For *gcc* and *g++*

```
$ sudo apt-get install build-essential
...
$ gcc --version
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
...
$ g++ --version
g++ (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
...
```

  - For *gfortran*

```
$ sudo apt-get install gfortran
...
$ gfortran --version
GNU Fortran (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
...
```

Installing

- Install SPEC CPU 2017 in three steps

  1) Mount the iso image

     - With root privileges (using *mount*)

```
$ mkdir /tmp/iso
$ sudo mount –o loop,ro cpu2017–1_0_5.iso /tmp/iso
```

     - Without root privileges (using *fuseiso*)

```
$ mkdir /tmp/iso
$ fuseiso /tmp/iso cpu2017–1_0_5.iso
```

Installing

- Install SPEC CPU 2017 in three steps

  2) Call the *install.sh* script and select the installation directory

```
$ cd /tmp/iso
$ mkdir ~/benchmarks
$ ./install.sh
SPEC CPU2017 Installation

Top of the CPU2017 tree is '/tmp/iso'
Enter the directory you wish to install to (e.g. /usr/cpu2017)
/home/andrei/benchmarks/SPEC17

Installing FROM /tmp/iso
Installing TO /home/andrei/benchmarks/SPEC17

Is this correct? (Please enter 'yes' or 'no')
yes
...
```

Installing

- Install SPEC CPU 2017 in three steps

  3) Unmount the iso image

     - With root privileges (using *umount*)

```
$ cd –
$ sudo umount /tmp/iso
$ rmdir /tmp/iso
```

     - Without root privileges (using *fusermount*)

```
$ cd –
$ fusermount –u /tmp/iso
$ rmdir /tmp/iso
```

Directory Structure

- The structure of the CPU 2017 directory tree is
  - **$SPEC or %SPEC%:** the root directory
    - **benchspec**: some suite-wide files
      - **CPU:** the benchmarks
    - **bin:** tools to run and report on the suite
    - **config:** config files
    - **Docs:** HTML and plaintext documentation
    - **result:** log files and reports
    - **tmp:** temporary files
    - **tools:** sources for the CPU 2017 tools

Directory Structure

- Within each of the individual benchmarks, the structure is
  - **nnn.benchmark:** root for this benchmark
    - **build**: benchmark binaries are built here
    - **data**
      - **all:** data used by all runs (if needed by the benchmark)
      - **ref:** the timed data set
      - **test:** data for a simple test that an executable is functional
      - **train:** data for feedback-directed optimization
    - **Docs:** documentation for this benchmark
    - **exe:** compiled versions of the benchmark
    - **run:** benchmarks are run here
    - **Spec:** SPEC metadata about the benchmark
    - **src:** The sources for the benchmark

Configuring

Configuring

- Configure SPEC CPU 17 in three steps

1) Copy base template files

```
$ cd ~/benchmarks/SPEC17
$ cp config/Example-gcc-linux-x86.cfg config/andrei.cfg
$ cp config/flags/gcc.xml config/flags/andrei-gcc.xml
```

Enter SPEC CPU 2017 directory.

Copy a base template file such as gcc for x86.

Unsupported compiler flags can be included for reportable runs.

Info: some configuration options can be overwritten at runtime.

Configuring

- Configure SPEC CPU 17 in three steps
  2) Add a flag (–no–pie) to the flags file (*andrei-gcc.xml)*

```
<flag name="F-mvis3"
   class="optimization">
   <![CDATA[<p>
      Generate code to take advantage of version 3 of the SPARC Visual Instruction Set extensions
   </p>]]>
</flag>

<flag name="F-no-pie"
   class="optimization">
   <![CDATA[<p>
      Do not generate position independent executable (PIE).
   </p>]]>
</flag>

<flag name="F-Ofast"
   class="optimization">
   <![CDATA[<p>
      Enable all optimizations of -O3 plus optimizations that are not valid for standard-compliant
      programs, such as re-ordering operations without regard to parentheses.
      <br /><a href="#gnote">Many more details are available</a>.
   </p>]]>
</flag>
```

New entry for flag –no–pie.

Configuring

- Configure SPEC CPU 17 in three steps
  3) Edit the config file (*andrei.cfg*)

```
...
#--------- Global Settings ---------
...
flagsurl             = $[top]/config/flags/andrei-gcc.xml
...
tune                 = base
...
#--------- How Many CPUs? ---------
...
intrate,fprate:
   copies            = 1
intspeed,fpspeed:
   threads           = 4
...
#--------- Compilers ---------
...
%ifndef %{gcc_dir}
%   define  gcc_dir        /usr
%endif
...
#--------- Baseline Tuning Flags ---------
...
default=base:
   OPTIMIZE       = -g -O3 -march=native -no-pie -fno-unsafe-math-optimizations -fno-tree-loop-vectorize
...
```

Use the new flags file.

Keep as *base* for now.

One instance for *intrate* and *fprate*.

Four threads for *intspeed* and *fpspeed*.

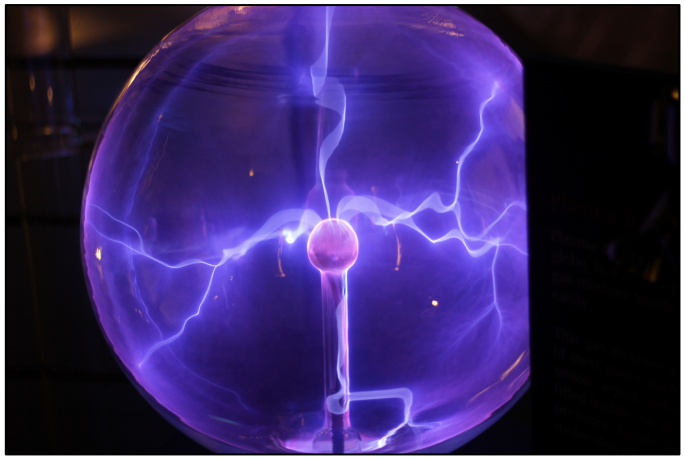
Path to *gcc*/*g++*/`gfortran`.

Add the `-no-pie` optimization flag for *base*.

Compilers Laboratory

DEPARTMENT OF COMPUTER SCIENCE
FEDERAL UNIVERSITY OF MINAS GERAIS, BRAZIL

Validating



Validating

Load simulation environment.

- Use runcpu♡ to run SPEC CPU 2017

Can use sets (ex.: *intrate, fprate*), benchmarks (ex.: *502.gcc_r*) or supersets (ex.: *specrate*, *specspeed* or *all*).

Reportable runs are suitable for public reporting

```
$ . shrc
$ runcpu --config=andrei.cfg --iterations=3 --reportable intrate
SPEC CPU(r) 2017 Benchmark Suites
Copyright 1995-2017 Standard Performance Evaluation Corporation (SPEC)


runcpu v5825
Using 'linux-x86_64' tools
Reading file manifests... read 32272 entries from 2 files in 0.22s (145963 files/s)
Loading runcpu modules.................
Locating benchmarks...found 47 benchmarks in 53 benchsets.
Reading config file '/home/andrei/benchmarks/SPEC17/config/andrei.cfg'
1 configuration selected:

 Action    Run Mode   Workload      Report Type
--------   --------   --------   -----------------
validate   rate       refrate    SPECrate2017_int
-------------------------------------------------------------------------------

Setting up environment for running intrate...
Starting runcpu for intrate...
...
```

Use our configuration file.

Valid runs must execute each benchmark at least three times.

Warning: `ulimit -s unlimited` may be required for 627.cam4_s (*fpspeed*)

The default action is *validate*.

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

Valid Results

- For reportable runs, a results report should look something like this

SPEC CPU2017 Integer Rate Result

My Corporation

SPECrate2017_int_base = 2.62

SPECrate2017_int_peak = Not Run

CPU2017 License: nnn (Your SPEC license number)
Test Sponsor: My Corporation
Tested by: My Corporation

Test Date: Dec-2021
Hardware Availability:
Software Availability:

Results Table

Benchmark	Base							Peak						
	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
500.perlbench_r	1	619	2.57	**618**	**2.57**	617	2.58							
502.gcc_r	1	413	3.43	412	3.43	**412**	**3.43**							
505.mcf_r	1	523	3.09	**522**	**3.09**	521	3.10							
520.omnetpp_r	1	608	2.16	602	2.18	**605**	**2.17**							
523.xalancbmk_r	1	618	1.71	600	1.76	**607**	**1.74**							
525.x264_r	1	657	2.66	**659**	**2.66**	659	2.66							
531.deepsjeng_r	1	476	2.41	**476**	**2.41**	475	2.41							
541.leela_r	1	**733**	**2.26**	735	2.25	733	2.26							
548.exchange2_r	1	487	5.38	**483**	**5.42**	483	5.42							
557.xz_r	1	**563**	**1.92**	563	1.92	564	1.92							

SPECrate2017_int_base = 2.62

SPECrate2017_int_peak = Not Run

Results appear in the order in which they were run. Bold underlined text indicates a median measurement.

General Notes

```
Environment variables set by runcpu before the start of the run:
LD_LIBRARY_PATH = "/usr/lib64/:/usr/lib/:/lib64"
```

Platform Notes

```
Sysinfo program /home/andrei/benchmarks/SPEC17/bin/sysinfo
Rev: r5974 of 2018-05-19 9bcde8f2999c33d61f64985e45859ea9
running on opencl Fri Dec  3 10:14:11 2021

SUT (System Under Test) info as seen by some common utilities.
For more information on this section, see
   https://www.spec.org/cpu2017/Docs/config.html#sysinfo

From /proc/cpuinfo
   model name : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
      1  "physical id"s (chips)
      12 "processors"
   cores, siblings (Caution: counting these is hw and system dependent. The following
   excerpts from /proc/cpuinfo might not be reliable.  Use with caution.)
      cpu cores : 6
      siblings  : 12
      physical 0: cores 0 1 2 3 4 5

From lscpu:
     Architecture:                    x86_64
     CPU op-mode(s):                  32-bit, 64-bit
```

(Continued on next page)

Invalid Results

- Otherwise, in case of any problems, an "Invalid Run" warning will be displayed in the report

Info: if the problems were caused by flags, the report can be recovered with `rawformat`.

SPEC CPU2017 Integer Rate Result

Copyright 2017-2021 Standard Performance Evaluation Corporation

My Corporation

SPECrate2017_int_base = 2.62

SPECrate2017_int_peak = Not Run

CPU2017 License: nnn (Your SPEC license number)
Test Sponsor: My Corporation
Tested by: My Corporation

Test Date: Dec-2021
Hardware Availability:
Software Availability:

Results Table

Benchmark	Base							Peak						
	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
500.perlbench_r	1	619	2.57	**618**	**2.57**	617	2.58							
502.gcc_r	1	413	3.43	412	3.43	**412**	**3.43**							
505.mcf_r	1	523	3.09	**522**	**3.09**	521	3.10							
520.omnetpp_r	1	608	2.16	602	2.18	**605**	**2.17**							
523.xalancbmk_r	1	618	1.71	600	1.76	**607**	**1.74**							
525.x264_r	1	657	2.66	**659**	**2.66**	659	2.66							
531.deepsjeng_r	1	476	2.41	**476**	**2.41**	475	2.41							
541.leela_r	1	**733**	**2.26**	735	2.25	733	2.26							
548.exchange2_r	1	487	5.38	**483**	**5.42**	483	5.42							
557.xz_r	1	**563**	**1.92**	563	1.92	564	1.92							

SPECrate2017_int_base = 2.62

SPECrate2017_int_peak = Not Run

Results appear in the order in which they were run. Bold underlined text indicates a median measurement.

General Notes

```
Environment variables set by runcpu before the start of the run:
LD_LIBRARY_PATH = "/usr/lib64/:/usr/lib/:/lib64"
```

Platform Notes

```
 Sysinfo program /home/andrei/benchmarks/SPEC17/bin/sysinfo
 Rev: r5974 of 2018-05-19 9bcde8f2999c33d61f64985e45859ea9
 running on opencl Fri Dec  3 10:14:11 2021

 SUT (System Under Test) info as seen by some common utilities.
 For more information on this section, see
    https://www.spec.org/cpu2017/Docs/config.html#sysinfo

 From /proc/cpuinfo
    model name : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
       1  "physical id"s (chips)
       12 "processors"
    cores, siblings (Caution: counting these is hw and system dependent. The following
    excerpts from /proc/cpuinfo might not be reliable.  Use with caution.)
       cpu cores : 6
       siblings  : 12
       physical 0: cores 0 1 2 3 4 5

 From lscpu:
      Architecture:                    x86_64
      CPU op-mode(s):                  32-bit, 64-bit
```

(Continued on next page)

Scripting the Results

- The results is generated in different formats, including a textual one that can be scripted and output in a CSV-like format

```
$ (echo "benchmark,run1,run2,run3"; \
   sed -n '/^---/,/^===/p' result/CPU2017.001.intrate.txt | \
   awk '$1 ~ /^[[:digit:]]{3}/ {bench[$1]=bench[$1] " " $3} \
        END { for (b in bench) print b bench[b]; }' | \
   tr " " "," | sort -n)
benchmark,run1,run2,run3
600.perlbench_s,615,617,616
602.gcc_s,780,778,783
605.mcf_s,1059,1061,1066
620.omnetpp_s,611,612,614
623.xalancbmk_s,588,602,588
625.x264_s,658,668,667
631.deepsjeng_s,579,580,579
641.leela_s,731,732,732
648.exchange2_s,485,487,513
657.xz_s,1281,1274,1276
```

Profiling

Profiling

- In the context of SPEC CPU 2017, we are interested in
  - **Dynamic profiling**: simulate a program monitoring its execution
    - Can use a Dynamic Binary Instrumentation (DBI) framework
      - Ex.: `valgrind`, `pin`, `DynamoRIO`, etc.
  - **Feedback-directed optimizations** (*FDO*): compiled in two phases
    - In the first phase, the compiler adds instrumentation to monitor a program execution
      - Ex.: `gcc -fprofile-generate ...`
    - In the second phase, the compiler is feed with information collected at runtime
      - Ex.: `gcc -fprofile-use ...`

Dynamic Profiling

- To perform dynamic profiling, we need to prepend a DBI we intend to use to the command that executes the benchmark
  - Ex.: `valgrind -q --tool=cfggrind`[♡] `--` *`benchmark args`*
- However, there is no native way of doing that, thus a patch[◊] was created to support such feature

```
$ cd ~/benchmarks/SPEC17
$ wget http://lac.dcc.ufmg.br/misc/spec2017-prepend.patch
$ patch -p1 < spec2017-prepend.patch
patching file MANIFEST
patching file bin/harness/benchmark.pm
patching file bin/harness/configpp
patching file bin/harness/parse.pl
patching file bin/harness/runcpu
```

- And now we can use the `--prepend` option

```
$ . shrc
$ runcpu --config=andrei.cfg --iterations=3 --reportable \
  --prepend="valgrind -q --tool=cfggrind --" intrate
```

♡: https://github.com/rimsa/CFGgrind ◊: http://lac.dcc.ufmg.br/misc/SPEC2017.html

Feedback-Directed Optimizations

- Let's use BOLT♤ to optimize a program with the following workflow

```
$ wget https://raw.githubusercontent.com/rimsa/CFGgrind/master/tests/test.c
$ gcc -g -O3 -no-pie -o test test.c -Wl,-q
$ for r in {1..3}; do \
  numbers=( $(seq 1 10000 | sort -R) ); \
  perf record -e cycles:u -j any,u -o run${r}.data -- ./test ${numbers[@]}; \
  perf2bolt -p run${r}.data -o run${r}.fdata ./test; \
done
$ merge-fdata run{1..3}.fdata > all.fdata
$ llvm-bolt ./test -o ./test.opt -report-stale -data=all.fdata \
  -reorder-blocks=cache+ -reorder-functions=hfsort -split-functions=2 \
  -split-all-cold -split-eh -dyno-stats
```

How can we apply this workflow to SPEC CPU 2017?

♤: https://github.com/facebookincubator/BOLT

Configuring

- Add `–Wl,–q` to the flags file (*andrei-gcc.xml)*

```
...
</flag>

<flag name="Wl-q"
   class="optimization"
   regexp="-Wl,-q">
   <example>-Wl,-q</example>
   <![CDATA[<p>
      Add the linker flag that enables realocation.
   </p>]]>
</flag>

<flag name="Wl-rpath"
   class="optimization"
   regexp="-Wl,-rpath,(\S+)(?=\s|$)"
   >
   <example>-Wl,-rpath,/path/to/lib</example>
   Add the specified directory to the runtime library search path used
   when linking an ELF executable with shared objects.
</flag>

...
```

New entry for flag –Wl,–q.

Configuring

- Update the config file (*andrei.cfg*)

```
...
#--------- Global Settings ---------
...
flagsurl             = $[top]/config/flags/andrei-gcc.xml
...
tune                 = peak
...
#--------- Peak Tuning Flags ---------
default=peak:
  # basepeak = yes  # if you develop some peak tuning, remove this line.
  OPTIMIZE = -g -O3 -march=native -no-pie -fno-unsafe-math-optimizations -fno-tree-loop-vectorize -Wl,-q

intrate,intspeed=peak: # flags for integer peak
   EXTRA_COPTIMIZE = -fno-strict-aliasing -fgnu89-inline
...
```

Change *tune* to *peak* (required for FDO).

Comment or remove this line.

Copy and edit this section from base to peak.

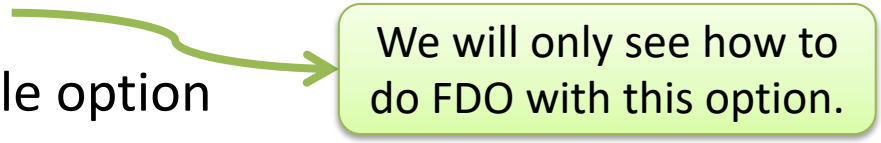
Copy *OPTIMIZE* option from *base* to *peak*.

And include `-Wl,-q` required for BOLT.

Feedback-Directed Optimizations

- SPEC CPU 2017 has builtin support for Feedback-directed optimizations♧ tipically using the following steps
  - Build a program with instrumentation
  - Run sample training workload to collect profile information
  - Apply the profile to produce a modified program

- This behavior can be controlled by
  - PASSn makevars
  - `fdo` shell options
  - *feedback* config file option
  - `--feedback` runcpu option



♧: https://www.spec.org/cpu2017/Docs/config.html#UsingFeedback

FDO Shell Options

- Some useful FDO shell options♔
  - **`fdo_pre0`**: Commands to execute before starting an FDO series
  - **`fdo_preN`**: Commands to be executed before pass N
  - **`fdo_pre_makeN`**: Commands to be done prior to Nth compile
  - **`fdo_make_passN`**: Commands to actually do the Nth compile
  - **`fdo_post_makeN`**: Commands to be done after the Nth compile
  - **`fdo_runN`**: Commands to be used for Nth training run
  - **`fdo_postN`**: Commands to be done at the end of pass N

Info: we can write shell scripts in these options.

We will work only with these options.

♔: https://www.spec.org/cpu2017/Docs/config.html#shellOptions

Curly Variables

- Some useful curly variables♧
  - **${baseexe}**: The first part of the executable name, which is <baseexe>_<tune>.<label>. For example, in "lbm_r_base.foo", baseexe is "lbm_r".
  - **${benchmark}**: The number and name of the benchmark currently being run, for example 519.lbm_r
  - **${benchname}**: The name of the benchmark currently being run, for example lbm_r
  - **${benchnum}**: The number of the benchmark currently being run, for example 519
  - **${command}**: The shell command line to run the current benchmark, for example ../run_base_test_sticky.0000/lbm_r_base.sticky 20 reference.dat 0 1 100_100_130_cf_a.of
  - **${commandexe}**: The executable for the current command, for example ../run_base_test_none.0000/lbm_r_base.sticky
  - **${iter}**: The current iteration number
  - **${tune}**: The tuning for the benchmark being run (base or peak)
  - **${workload}**: The current workload number (within the iteration)N

♧: https://www.spec.org/cpu2017/Docs/config.html#sectionI.F.5

Configuring

- Update the config file (*andrei.cfg*) to include FDO

```
#--------- Peak Tuning Flags ---------
default=peak:
   basepeak = yes  # if you develop some peak tuning, remove this line.

fdo_pre0 = rm -rf ${benchmark}.data ${benchmark}.fdata; \\
           touch ${benchmark}.fdata


fdo_run1 = mv ${benchmark}.fdata previous.fdata; \\
           perf record -e cycles:u -j any,u -o ${benchmark}.data -- ${command}; \\
           perf2bolt -p ${benchmark}.data -o current.fdata ${baseexe}; \\
           merge-fdata previous.fdata current.fdata > ${benchmark}.fdata; \\
           rm -rf ${benchmark}.data previous.fdata current.fdata


fdo_post1 = if [ -s "${benchmark}.fdata" ]; then \\
                mv ${baseexe} ${baseexe}-orig; \\
                llvm-bolt ${baseexe}-orig -o ${baseexe} -report-stale -data=${benchmark}.fdata
-reorder-blocks=cache+ -reorder-functions=hfsort -split-functions=2 -split-all-cold -split-eh
-dyno-stats; \\
            fi

...
```

Clean up previous runs and create empty fdata using `${benchmark}`.

Profile `${command}` with perf and convert output to fdata.

Optimize `${baseexe}` with `llvm-bolt`.

Debugging

- Debug FDO using ––fake.

```
$ runcpu --config=andrei.cfg --fake 502.gcc_r
...
%% Fake commands from fdo_pre0 (rm -rf ${benchmark}.data ${benchmar...):
rm -rf 502.gcc_r.data 502.gcc_r.fdata;  touch 502.gcc_r.fdata
%% End of fake output from fdo_pre0 (rm -rf ${benchmark}.data ${benchmar...)
...
# Starting run for copy #0
cd /home/andrei/benchmarks/SPEC17/benchspec/CPU/502.gcc_r/build/build_peak_mytest-m64.0000
mv 502.gcc_r.fdata previous.fdata;  perf record -e cycles:u -j any,u -o 502.gcc_r.data --
../build_peak_mytest-m64.0000/cpugcc_r 200.c -O3 -finline-limit=50000 -o 200.opts-O3_-
finline-limit_50000.s > 200.opts-O3_-finline-limit_50000.out 2>> 200.opts-O3_-finline-
limit_50000.err;  perf2bolt -p 502.gcc_r.data -o current.fdata cpugcc_r;  merge-fdata
previous.fdata current.fdata > 502.gcc_r.fdata;  rm -rf 502.gcc_r.data previous.fdata
current.fdata
...
%% End of fake output from benchmark_run (/home/andrei/benchmarks/SPEC17/bin/speci...)
...
%% Fake commands from fdo_post1 (if [ -s "${benchmark}.fdata" ]; the...):
if [ -s "502.gcc_r.fdata" ]; then  mv cpugcc_r cpugcc_r-orig;  llvm-bolt cpugcc_r-orig -o
cpugcc_r -report-stale -data=502.gcc_r.fdata -reorder-blocks=cache+ -reorder-
functions=hfsort -split-functions=2 -split-all-cold -split-eh -dyno-stats;  fi
%% End of fake output from fdo_post1 (if [ -s "${benchmark}.fdata" ]; the...)
...
```

fdo_pre0

fdo_run1

fdo_post1

Executing

- If everything is alright, execute the benchmark

```
$ runcpu --config=andrei.cfg \
  --iterations=3 --reportable \
  intrate
```

SPEC CPU2017 Integer Rate Result

My Corporation

SPECrate2017_int_base = 2.62

SPECrate2017_int_peak = 2.67

CPU2017 License: nnn (Your SPEC license number)
Test Sponsor: My Corporation
Tested by: My Corporation

Test Date: Dec-2021
Hardware Availability:
Software Availability:

Results Table

Benchmark	Base							Peak						
	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
500.perlbench_r	1	616	2.59	**616**	**2.59**	618	2.58	1	**576**	**2.76**	578	2.76	574	2.77
502.gcc_r	1	412	3.43	**412**	**3.44**	410	3.45	1	**389**	**3.64**	388	3.65	389	3.64
505.mcf_r	1	523	3.09	**523**	**3.09**	520	3.11	1	**524**	**3.08**	525	3.08	523	3.09
520.omnetpp_r	1	612	2.14	**607**	**2.16**	604	2.17	1	598	2.19	601	2.18	**598**	**2.19**
523.xalancbmk_r	1	**608**	**1.74**	610	1.73	605	1.75	1	596	1.77	604	1.75	**601**	**1.76**
525.x264_r	1	659	2.66	659	2.66	**659**	**2.66**	1	649	2.70	**650**	**2.69**	651	2.69
531.deepsjeng_r	1	475	2.41	**475**	**2.41**	474	2.42	1	471	2.43	**471**	**2.43**	471	2.43
541.leela_r	1	732	2.26	733	2.26	**733**	**2.26**	1	734	2.26	**732**	**2.26**	732	2.26
548.exchange2_r	1	490	5.35	**489**	**5.36**	485	5.41	1	492	5.32	**491**	**5.33**	490	5.35
557.xz_r	1	562	1.92	**563**	**1.92**	563	1.92	1	558	1.94	561	1.92	**558**	**1.93**

SPECrate2017_int_base = 2.62

SPECrate2017_int_peak = 2.67

Results appear in the order in which they were run. Bold underlined text indicates a median measurement.

General Notes

```
Environment variables set by runcpu before the start of the run:
LD_LIBRARY_PATH = "/usr/lib64/:/usr/lib/:/lib64"
```

Platform Notes

```
Sysinfo program /home/andrei/benchmarks/SPEC17/bin/sysinfo
Rev: r5974 of 2018-05-19 9bcde8f2999c33d61f64985e45859ea9
running on opencl Thu Dec  2 22:59:43 2021

SUT (System Under Test) info as seen by some common utilities.
For more information on this section, see
   https://www.spec.org/cpu2017/Docs/config.html#sysinfo

From /proc/cpuinfo
   model name : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
      1  "physical id"s (chips)
      12 "processors"
   cores, siblings (Caution: counting these is hw and system dependent. The following
   excerpts from /proc/cpuinfo might not be reliable.  Use with caution.)
      cpu cores : 6
      siblings  : 12
      physical 0: cores 0 1 2 3 4 5

From lscpu:
     Architecture:                    x86_64
     CPU op-mode(s):                  32-bit, 64-bit
```

(Continued on next page)

Questions?