Programmatically run SPEC CPU2017

D. Giordano (CERN)

pre-GDB Benchmarking WG

12 June 2018

SPEC CPU 2017 and HEPiX Benchmarking WG

SPEC CPU2017 (**SC17**) is available since ~1 year

Plan to study:

- Features of the SC17 suite: benchmark mix, compiler flags, rate Vs speed
- Relationship w.r.t. HEP job mix

But so far only scattered results produced by our community

- Main results presented today

IMHO the main reason is the difficulty in:

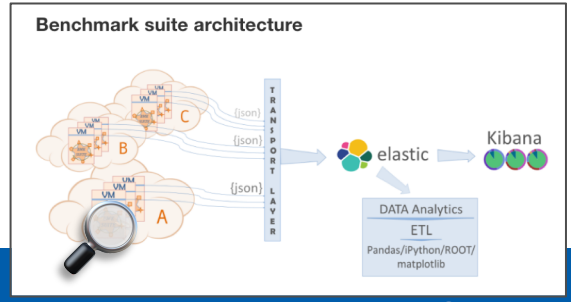
- Defining the appropriate configuration(s)
- Performing reproducible studies: collecting, sharing and tracking results

In the next slides a possible solution to those obstacles is proposed

“Push the button” & run SC17

Ability to trigger SC17 runs and extract results using a single wrapper script call

- Pass arguments via command line
  - `cern-benchmark --benchmarks=spec2017 --spec2017_path=<SC17 installation dir> --spec2017_iter=<num iterations> --spec2017_bmk=<some bset. Eg 511.povray_r>`
- Install the software from tarball archive [optional]
  - `cern-benchmark --benchmarks=spec2017 […] --spec2017_url=<url to SC17 tarball to install>`
- Run in sequence with other benchmarks
  - `cern-benchmark --benchmarks="hs06_32;whetstone;DB12;kv;hs06_64;spec2017" […]`
- Collect results of each running benchmark in a json document
  - Simplify portability and usage in other applications, e.g.: monitoring and analytics



A wrapper script similar to HS06

The wrapper to trigger SC17 is very similar to the HS06 script (runspec.sh)

- Evaluate the score as the geometric mean of the independent runs
  - Same formula as HS06
- NB: standalone script, can be used independently of the full suite

Optional arguments include:

- *Config. file*
  - Adapted from the SPEC CPU template Example-gcc-linux-x86.cfg
  - NB: Currently sets `OPTIMIZE = -g -O3` will be changed to `-O3 -fPIC -pthread` (see M. Alef talk)
- *Set of benchmarks*
  - Defined a new set (`pure_rate_cpp.bset`) including only pure C++/C benchmarks as in HS06

https://www.spec.org/cpu2017/Docs/index.html#benchmarks

The Benchmarks

SPEC CPU2017 has 43 benchmarks, organized into 4 suites:

```
SPECrate 2017 Integer          SPECspeed 2017 Integer
SPECrate 2017 Floating Point   SPECspeed 2017 Floating Point
```

Benchmark pairs shown as:

```
5nn.benchmark_r / 6nn.benchmark_s
```

are similar to each other. Differences include: compile flags; workload sizes; and run rules. See: [OpenMP] [memory] [rules]

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 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

Bmk HS06
444.namd
447.dealII
450.soplex
453.povray
471.omnetpp
473.astar
483.xalancbmk

this set defines pure_rate_cpp.bset

similarity with HS06

How to install the benchmarking suite

On SLC6

- Install from benchmarking suite gitlab repository

On CC7

- Follow the SLC6 approach, OR
- Use **docker container** based on SLC6 image
  - Published in gitlab registry
    - `docker run --net=host -v /tmp:/tmp -v /var/HEPSPEC:/var/HEPSPEC gitlab-registry.cern.ch/cloud-infrastructure/cloud-benchmark-suite/cloud-benchmark-suite-standalone:1.5 cern-benchmark […]`

Upcoming

- New rpm, to be made available in the CERN SLC6/7 repos
- New documentation, describing all those features

The profile report


```
"message": {
  "host": {...},
  "_id": "bmk14-slc6-w54ynvdxsu_5dde88ba-b6b2
  "_timestamp_end": "2018-06-11T03:03:16Z",
  "profiles": {
    "whetstone": {...},
    "hs06_64": {...},
    "skv": {...},
    "DB12": {...},
    "hs06_32": {...},
    "spec2017": {...}
  },
  "_timestamp": "2018-06-10T16:30:03Z"
```


Host metadata


```
"host": {
  "benchmark_target": "machine",
  "meminfo": 60515988,
  "UID": "bmk14-slc6-w54ynvdxsu_5dde88ba-b6b2-4f3f-a776-74901
  "classification": "i6_14_f6m79s1_mhz2394.454",
  "freetext": "VM-14-SLC6",
  "cpuname": "Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz",
  "ip": "188.184.189.232",
  "osdist": "Scientific Linux CERN SLC release 6.9 (Carbon)",
  "bogomips": 4788.9,
  "VO": "continuous-bmk",
  "cpunum": 14,
  "pnode": "i69234828294880",
  "mp_num": 14,
  "pyver": "2.6.6",
  "cloud": "CERN-gva_project_035"
},
```


SC17 report, including config options


```
"spec2017": {
  "extra_coptimize": " -fno-strict-aliasing",
  "end": "Mon Jun 11 05:02:24 CEST 2018",
  "bmks": {...},
  "start": "Mon Jun 11 02:35:52 CEST 2018",
  "score": 21.58,
  "num_bmks": 8,
  "runcpu_args": "/var/HEPSPEC/SPEC2017/bin/harness/runcpu --configfile cern-gcc-linux-x86.cfg --nobuild --noreportable --iterations 1 --nore
-x86.cfg --nobuild --noreportable --iterations 1 --noreportable --nopower --runmode rate --tune base --size refrate 520.omnetpp_r 523.xalancbmk_r 531
  "avg_core_score": 1.541,
  "optimize": " -g -O3",
  "bset": "pure_rate_cpp",
  "compiler": " C/C++/Fortran: Version 6.2.0 of GCC, the GNU Compiler Collection "
}
},
```


SC17 individual benchmark scores

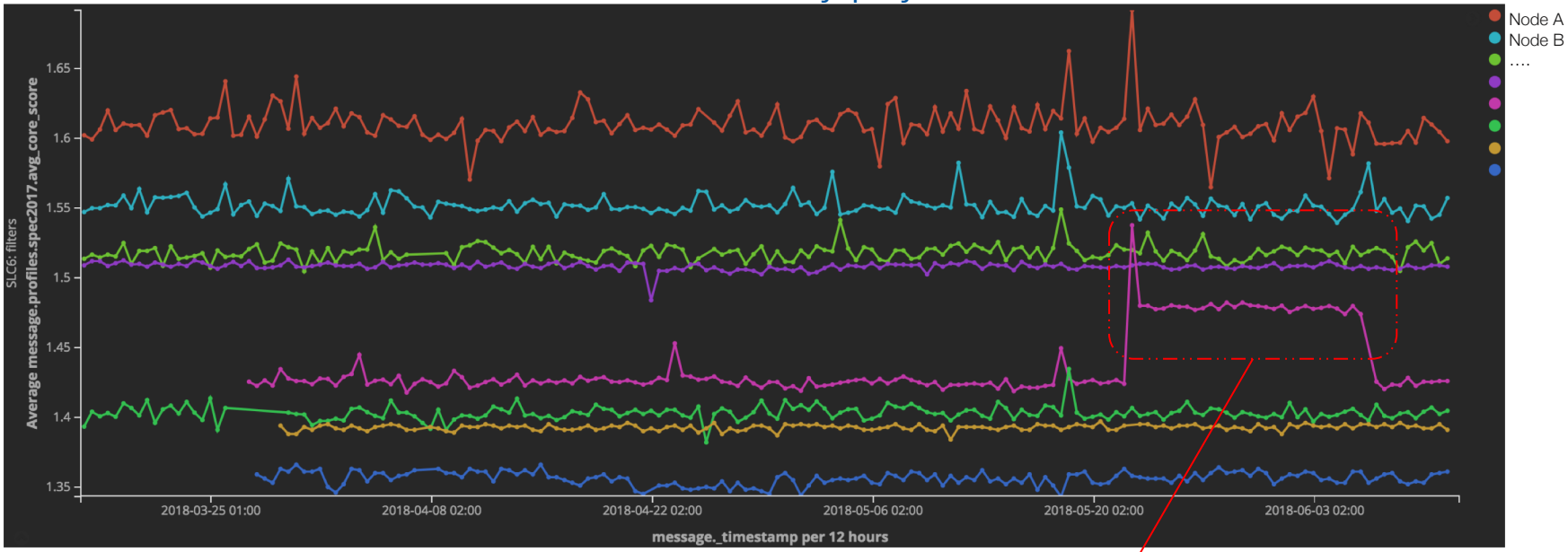

```
"bmks": {
  "450.soplex": [...],
  "471.omnetpp": [...],
  "447.dealII": [...],
  "473.astar": [...],
  "444.namd": [...],
  "453.povray": [...],
  "483.xalancbmk": [...]
```

```
1.62, 1.61, 1.62, 1.63, 1.61, 1.62, 1.61, 1.61, 1.61, 1.62, 1.61, 1.62, 1.6, 1.62
1.22, 1.22, 1.22, 1.23, 1.22, 1.22, 1.22, 1.22, 1.22, 1.22, 1.22, 1.22, 1.23, 1.22
2.17, 2.15, 2.16, 2.16, 2.17, 2.16, 2.14, 2.17, 2.13, 2.18, 2.14, 2.16, 2.15, 2.19
1.02, 1.01, 1.01, 1.21, 1.01, 1.01, 1.01, 1, 1.01, 1.01, 1.01, 1.01, 1, 1.16
1.04, 1.04, 1.03, 1.04, 1.04, 1.05, 1.04, 1.04, 1.06, 1.01, 1.04, 1.05, 1.04, 1.02
2.34, 2.32, 2.32, 2.32, 2.35, 2.32, 2.33, 2.34, 2.33, 2.32, 2.34, 2.35, 2.33, 2.33
1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.69, 1.7, 1.7, 1.68, 1.69, 1.7, 1.7, 1.71
1.76, 1.77, 1.78, 1.74, 1.76, 1.77, 1.79, 1.77, 1.77, 1.79, 1.77, 1.76, 1.77, 1.73
```


At CERN this json doc is transferred to ElasticSearch via a message broker

The last 3 months of data

SC17 score/core by physical node



Example of start time desynchronization between VMs on the same physical node (*cleaned in the following analysis*)

Analysis' results

- Description of the benchmarking setup @ CERN

- Focus on two questions
  1. How much SC17 is different from HS06?
  2. Are all the individual benchmarks in the SC17 independent?

     i.e. Are all benchmarks needed to build a representative benchmark mix?

Amount of data collected

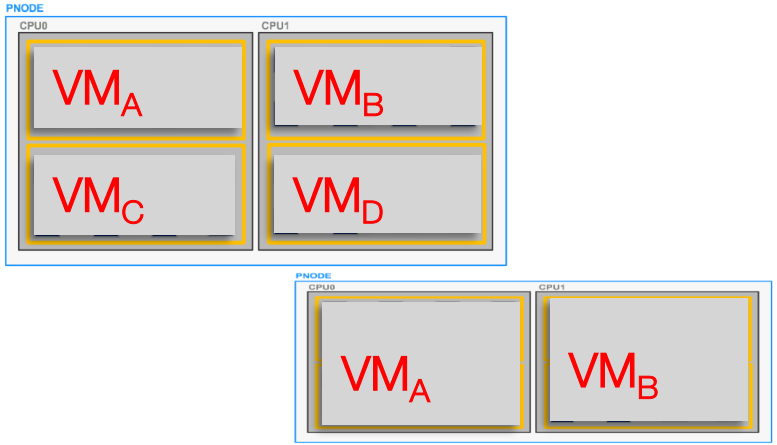
cpuname	count
Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz	1097.0
Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz	278.0
Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz	486.0
Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz	476.0
Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz	572.0
Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz	1005.0
Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz	995.0

Benchmarking approach

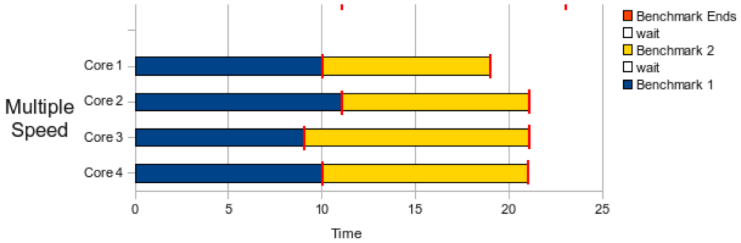
- Profile full physical node, SMT enabled
- Split node resources in VMs.
  No overcommit, NUMA aware flavors
- Benchmark suite run synchronously in all VMs
- Benchmarks sequence

```
--benchmarks="hs06_32;whetstone;DB12;kv;hs06_64;spec2017”
```

- Run `a la HS06`
  - use N x SPECrate to mimic the HS06 multiple speed
- NB: running on VMs gives an overhead of few % respect to bare metal
  - Presented results are *ratios*, mainly cancel this effect



HS06 running mode



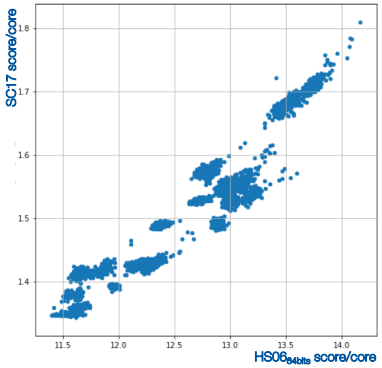
https://w3.hepix.org/benchmarking/how_to_run_hs06.html

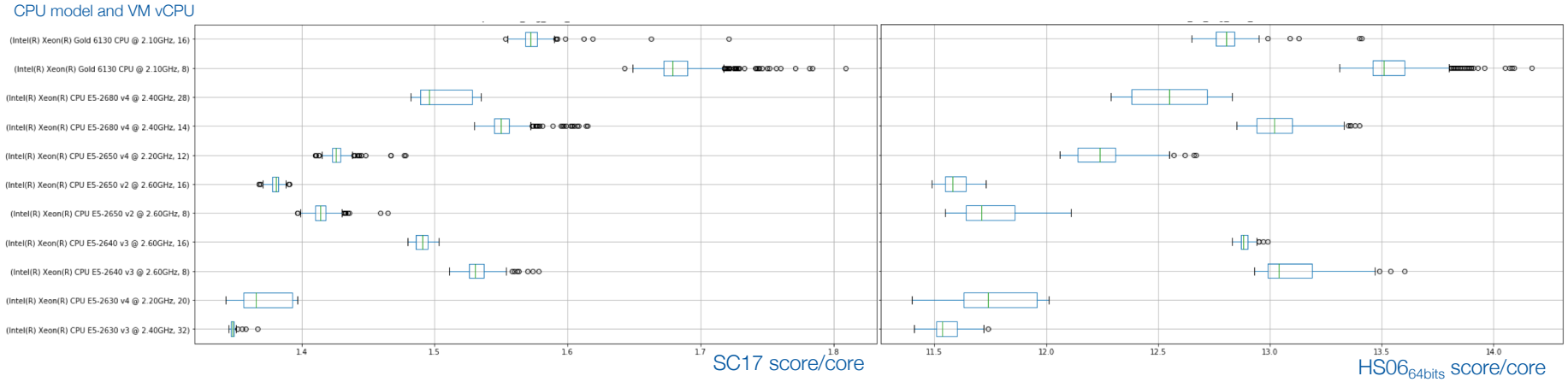
Are SC17 and $HS06_{64bit}$ correlated?

Implications

- No correlation => weird situation
- High correlation => questioning the change

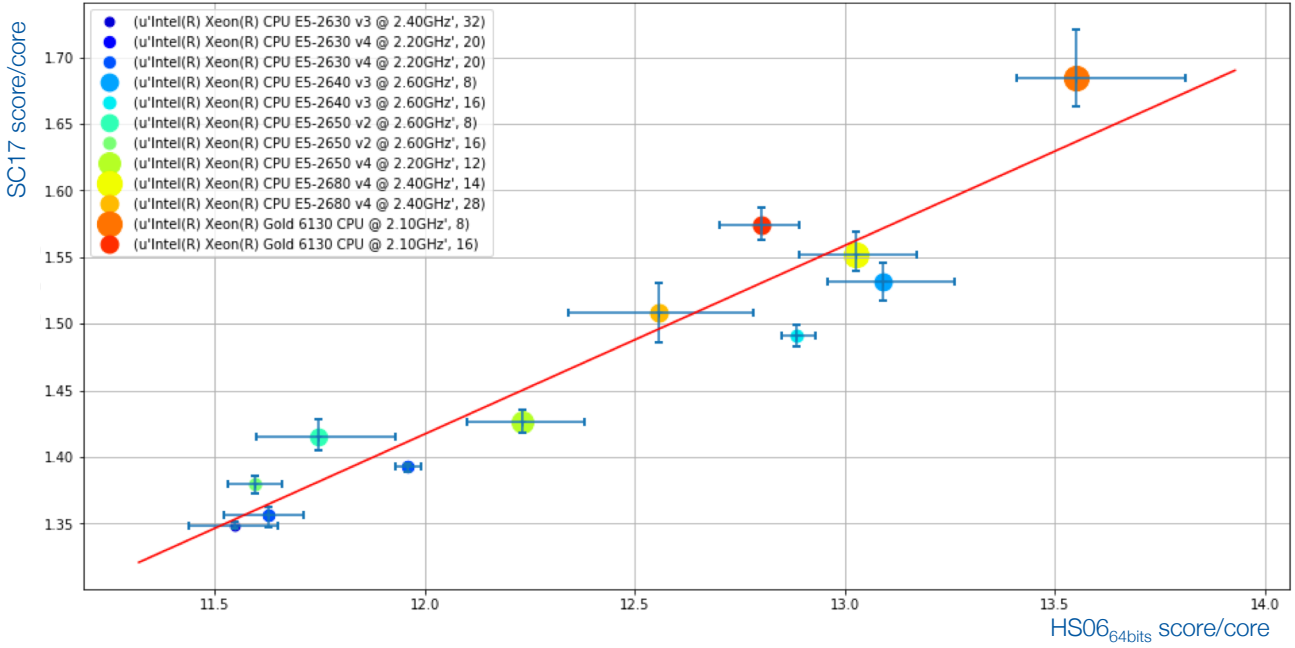
Reminder: using just *pure_rate_cpp*





Are SC17 and $HS06_{64bit}$ correlated?

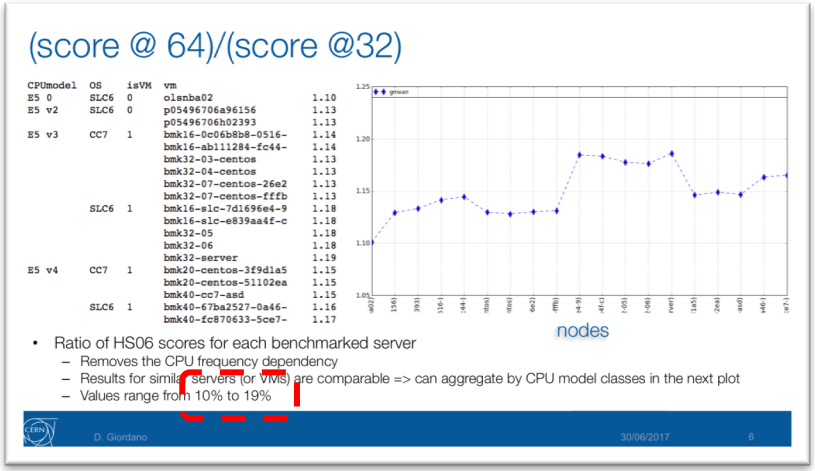
- Error bars represent the percentiles [5%,95%]
- Marker size proportional to the number of entries
- Correlation coefficient of the mean values: 0.950
- Linear fit
  SC17 = m * HS06 + q
  - m = 0.141, q= -0.280
  - Residuals ratios: (extr./meas.-1) < 4%

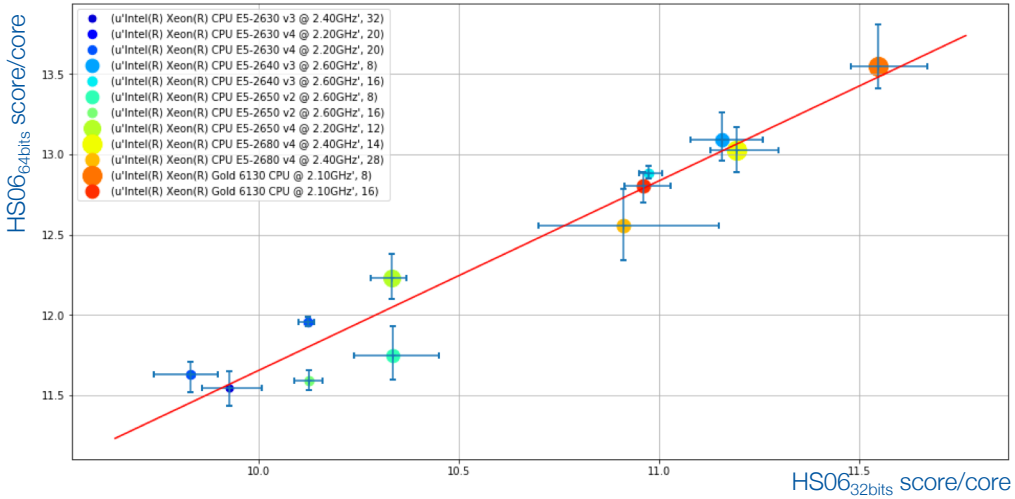


Validation: HS06 32bits Vs 64bits

Study on HS06 reported in the past

- Discrepancy between HS06 @ 32bits and 64bits is ~10%





- Correlation coefficient of the mean values: 0.97
- Linear fit
  $HS06_{64} = m * HS06_{32} + q$
  - m = 1.18, q= -0.156
  - Residuals ratios: (extr./meas.-1) < 3%

Are the individual SC17 benchmarks independent?

Are all SC17 benchmarks (`pure_rate_cpp`) needed?

- Less benchmarks => Shorter runtime
  - Currently a run of `pure_rate_cpp` (8 benchmarks) takes <2.5hour/iteration> in the 7 tested CPU models

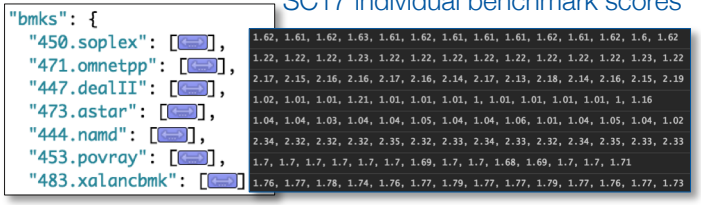
- Better control of SC17 score Vs HEP job mix

- Studied if a subset of the `pure_rate_cpp` is as well representative of the performance score

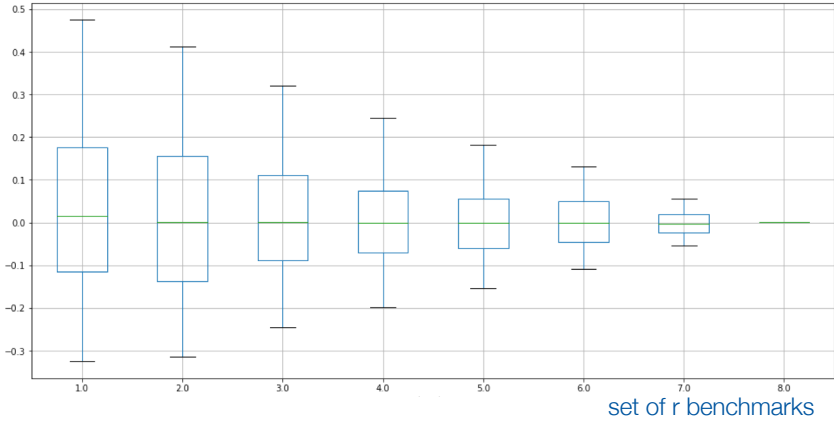
Approach

- Build all *r* combinations of the 8 benchmarks
  - With *r* in [1,8], number of combination = *C(8,r)*
- For each combination re-evaluate the SC17 score
  - NB: The benchmarking suite stores the individual results of each benchmark and thread
- Aggregate the scores by CPU model, compute stats (weighted avg and std, min, max)
- Measure the residuals ratio respect to the original SC17 (*C(8,8)*)

SC17 individual benchmark scores

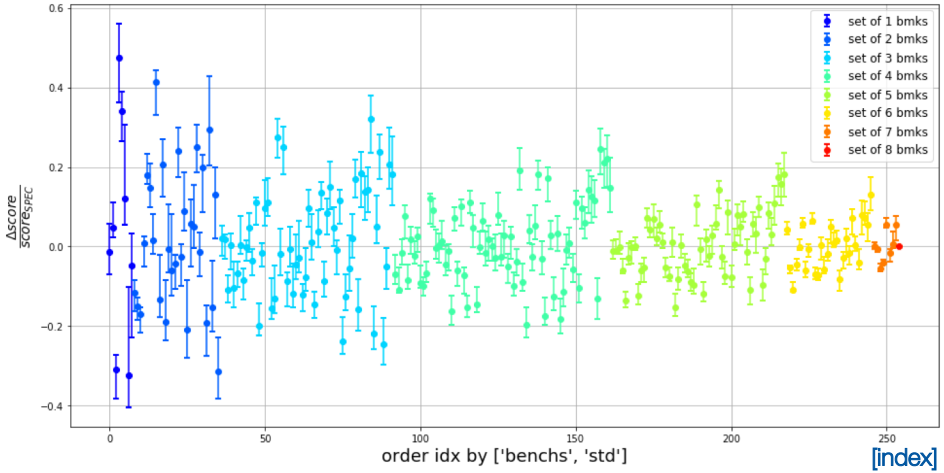


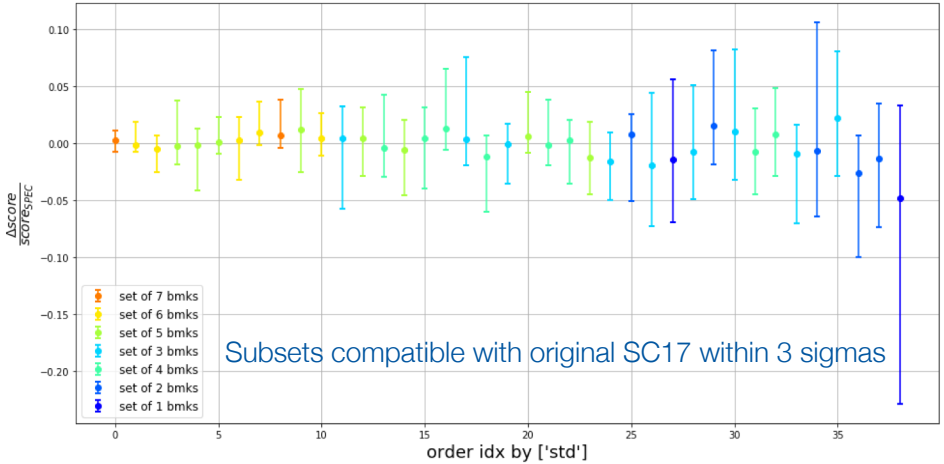
Mean residual ratio for each CPU model



Computed scores

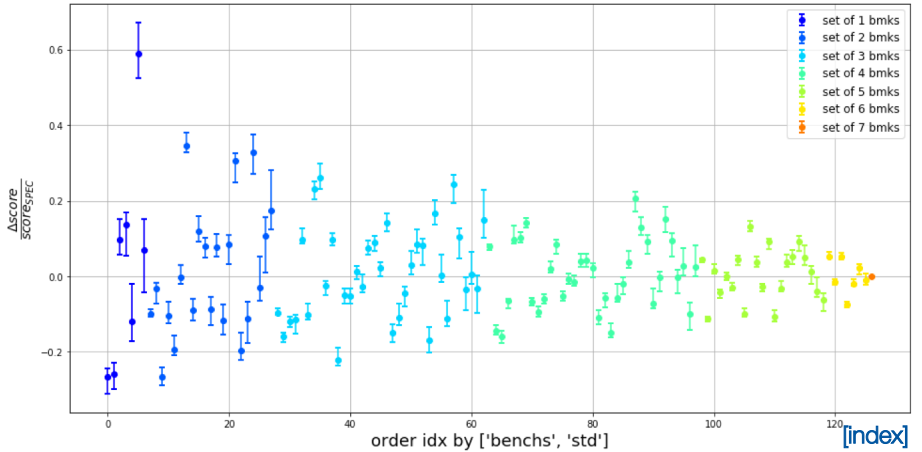
- New scores discrepancy ranges from <1% to >50%
- NB: Error bars represent the [min,max]
- Several subset with only 2 or 3 benchmarks are well in agreement with the original SC17 score
- Example: compatible with the original SC17 within 1 sigma
  - 508.namd_r , 510.parest_r, 531.deepsjeng_r
  - Discrepancy
    - max= 0.017, min= -0.035
    - mean= -0.001 ± 0.005

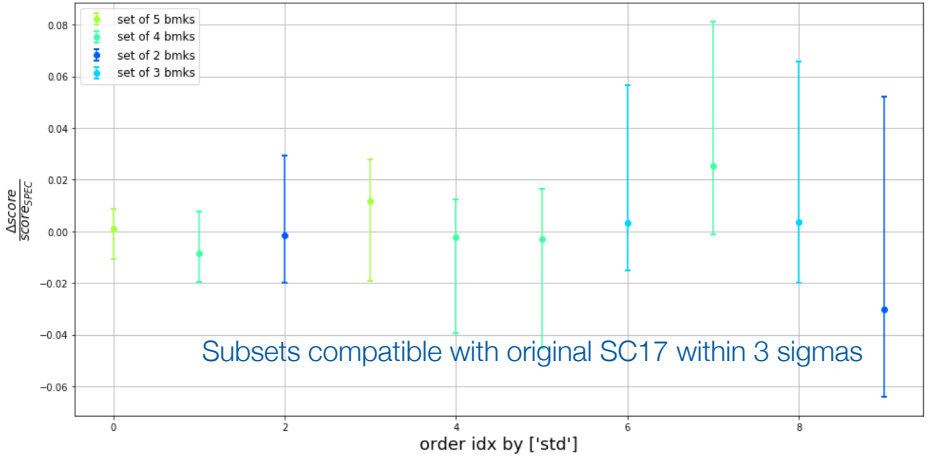




What about HS06?

- Similar behavior, but larger incompatibilities
- NB: Error bars represent the [min,max]
- Examples of reduced set compatible with the original HS06 within 1 sigma
  - 444.namd, 447.dealII, 471.omnetpp
    - Discrepancy
      - max= 0.056, min= -0.015
      - mean= -0.003 ± 0.007
  - 444.namd, 447.dealII, 473.astar
    - Discrepancy
      - max= 0.066, min= -0.02
      - mean= -0.004 ± 0.009





Summary

Convergence toward a common tool (and configuration) to run SPEC CPU2017 (SC17)

- Now possible using a wrapper script in the style of HS06 runspec.sh
- Additional features offered by the benchmarking suite
  - Documentation will be available in the next weeks

Correlation between SC17 and HS06

- Very high correlation, measured on 7 different Intel CPU models
  - Covering >90% of the CERN Openstack cloud
- Need to extend the study to other CPU models and architectures
- If the very high correlation remains, why changing from HS06 to SC17?

The individual SC17 benchmarks (`pure_rate_cpp`) are not fully independent

- It is possible to find subsets of benchmarks giving the same score ratio
- How those set will scale with the HEP job mix? To be studied

CERN