Detecting/Analyzing Unstable Performance Behavior

Cédric Valensi, Emmanuel Oseret, Hugo Bolloré, Mathieu Tribalat, Kevin Camus, William Jalby (UVSQ/ECR/University Paris Saclay)
cedric.valensi@uvsq.fr , emmanuel.oseret@uvsq.fr, william.jalby@uvsq.fr, mathieu.tribalat@uvsq.fr, kevin.camus@uvsq.fr

http://www.maqao.org

With ECR, INTEL, CEA, SiPearl, ATOS and UVSQ support

TYPICAL ISSUES

- Between runs, execution times differ widely: STABILITY
- Performance measurements have been carried out but are they meaningful ? QUALITY/RELIABILITY
- The same loop is executed billions of times, do all of instances have a similar behaviour ? VARIABILITY

We will review each issue and present how MAQAO/OV is handling it.

Problems listed above have a wide range of applicability (from unicore all of the way to full multinode systems). In this talk, we will focus on single node cases.

- MAQAO is a performance analyzis and optimisation framework operating at binary level developed at UVSQ since 2004
  - Complementary modules, each of them focusing on one aspect of performance analysis: profiler, static analyzer, simple simulators, value profiler, decremental analyzer, …
  - Support for Intel/AMD x86-64 and ARM (ongoing)
  - http://www.maqao.org

- ONE View: Performance View Aggregator module
  - Goal: Guiding the user through the analysis & optimization process
  - Automatizes execution of experiments invoking other MAQAO modules and aggregates their results to produce high-level reports in HTML or XLSX format

UNSTABILITIES BETWEEN DIFFERENT EXECUTIONS

UNSTABILITIES BETWEEN DIFFERENT EXECUTIONS

DETECTION

- Needs multiple runs and to have meaningful statistics, several tens of runs needed, typically at least more than 30…
- Costly but necessary: unless you perform multiple runs, you have no idea of potential unstabilities.

ANALYSIS

- Statistics on total execution times are interesting but not enough
- More detailed statistics are needed

UNSTABILITIES BETWEEN DIFFERENT EXECUTIONS

MAQAO/OV approach. We provide a dedicated operating mode which automatically:

- Manages and launch K runs separated by P seconds: K and P being parameters set by the user
- Performs statistical analysis of the K global execution times.
- Performs comparative and statistical analysis at the function level. Fundamental to pinpoint the guilty function(s) creating unstabilities.

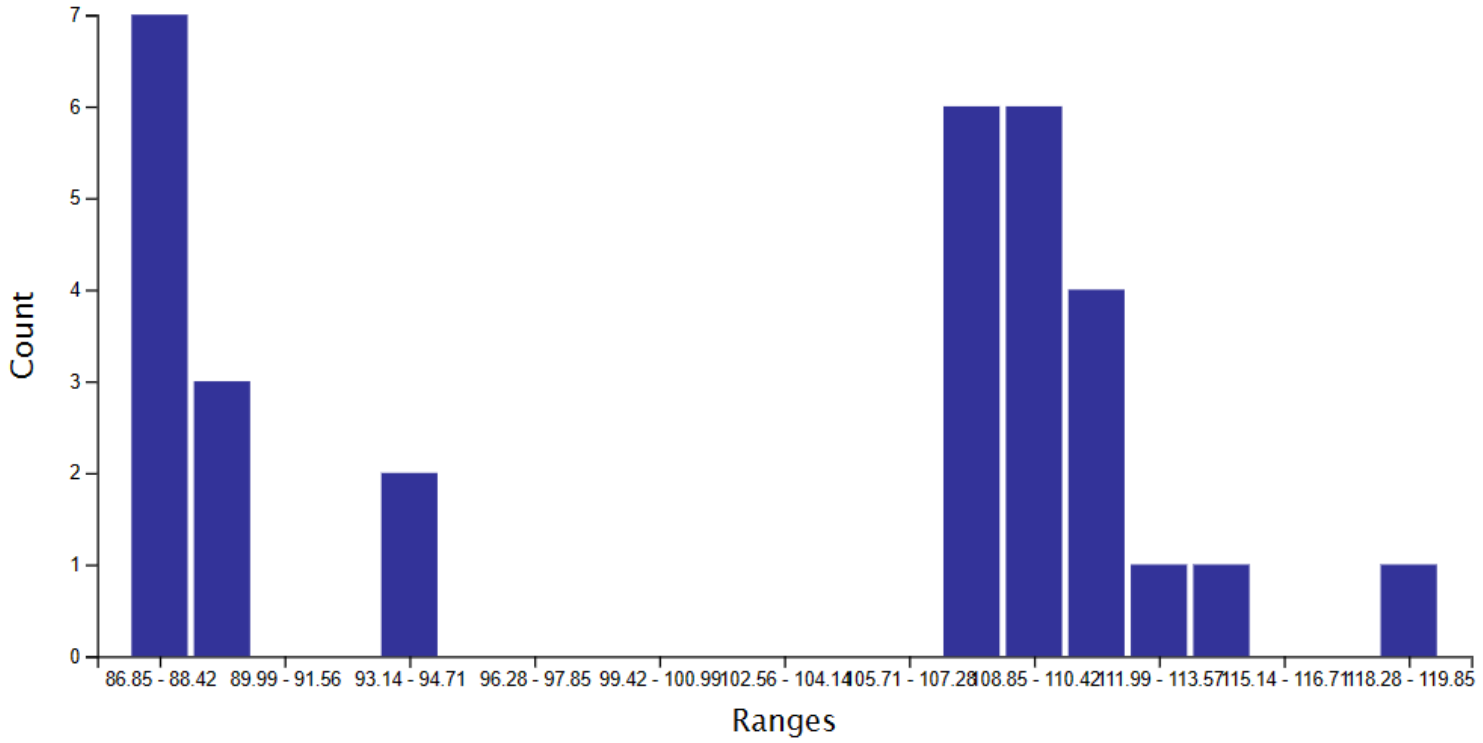
ISOBINARY: STABILITY RUNS (1)

GROMACS running on a 128 cores EPYC2 (ROME/ZEN2)
31 runs

Profiled Time

min	med	avg	max
86.85	108.72	102.11	118.28

Percentile Index	10	20	30	40	50	60	70	80	90	100
Value	87.23	87.57	88.81	94.69	108.69	108.82	109.18	110.39	111.41	118.28



ISOBINARY: STABILITY RUNS (2)

GROMACS running on a 128 cores EPYC2 (ROME/ZEN2)
31 runs

▶ Colums Filter

Name	Module	Time w.r.t. Wall Time (s)	min (Time w.r.t. Wall Time) (s)	avg (Time w.r.t. Wall Time) (s)	med (Time w.r.t. Wall Time) (s)	max (Time w.r.t. Wall Time) (s)
○ _mpc_omp_callback_run	libmpcomp.so.0.0.0	17.25	12.37	15.94	17.35	18.33
○ _mpc_thread_ethread_mxn_engine_func_kernel_thread	libmpcthread.so.0.0.0	15.79	14.79	15.70	15.72	17.22
▶ void _INTERNAL7040d793::clearBufferFlagged<3>(nbnxn_atomdata_t const&, int, gmx::ArrayRef)	libgromacs_mpi.so.8.0.0	11.72	1.97	7.80	11.34	12.6
○ _mpc_common_spinlock_trylock	libmpclaunch.so.0.0.0	8.34	5.86	7.46	7.59	8.8
▶ fft5d_execute(fft5d_plan_t*, int, gmx_wallcycle*)	libgromacs_mpi.so.8.0.0	5.5	5.49	6.47	5.73	11.66
▶ nbnxn_atomdata_add_nbat_f_to_f_reduce#0xcb8665	libgromacs_mpi.so.8.0.0	5.08	0.84	3.36	4.89	5.4

Min, avg, med and max are computed over the 31 runs.

With that info, The issue could be tracked down to a memory leak in the runtime library.

MEASUREMENT « QUALITY »

How much can the user trust our recommendations ??

- **Measurement intrusiveness can severely pollute measurement:** probe code could be heavy and distort measurement but also gathering too much measurement data can perturb cache behavior
- **When using sampling, too few samples might be the sign of low "quality" and unstable measurement.**
- **When using tracing, measuring too short durations within an OoO machine might lead to low level quality measurements:** any duration measured under 500 cycles is subject to caution. Any duration measured under 100 cycles is close to noise.

MEASUREMENT INTRUSIVENESS

- All of MAQAO/OV instrumentation are performed at the binary level : we analyse what we are running
- Probe code is carefully designed in order to limit performance impact
- Probe overhead is evaluated and correction are performed on the measurements carried out.
- Try to limit amount of data gathered: select carefully loops to be instrumented, and for example, for most of measurement no timestamps are collected (and therefore such a restriction prevents timelines production).

SAMPLING INSTRUMENTATION QUALITY

- Any element (function, loop) with too few samples is flagged to the user with simple color code: measurements corresponding to less than 100 (resp. 300) samples are displayed in red (resp. orange) cells.

TRACING INSTRUMENTATION QUALITY

- Similar strategy as for sampling quality: report directly to the user cases with durations less than 200 cycles.
- Rely on multiple execution of the same loops and measurement variability across instances (more details in the last part of the talk).

PERFORMANCE VARIATIONS ACROSS MULTIPLE LOOP INSTANCES WITHIN A SAME RUN

PERFORMANCE VARIATIONS

TWO ISSUES

DETECTION

- Are execution times more or less constant across loop instances
- If not, how are they distributed

ANALYSIS

- Can we exploit performance variations for optimization
- Is variation correlated with loop iteration count?
- Is variation correlated with call sites ? Or other loop parameter ?

MAQAO/ONE VIEW APPROACH: DETECTION

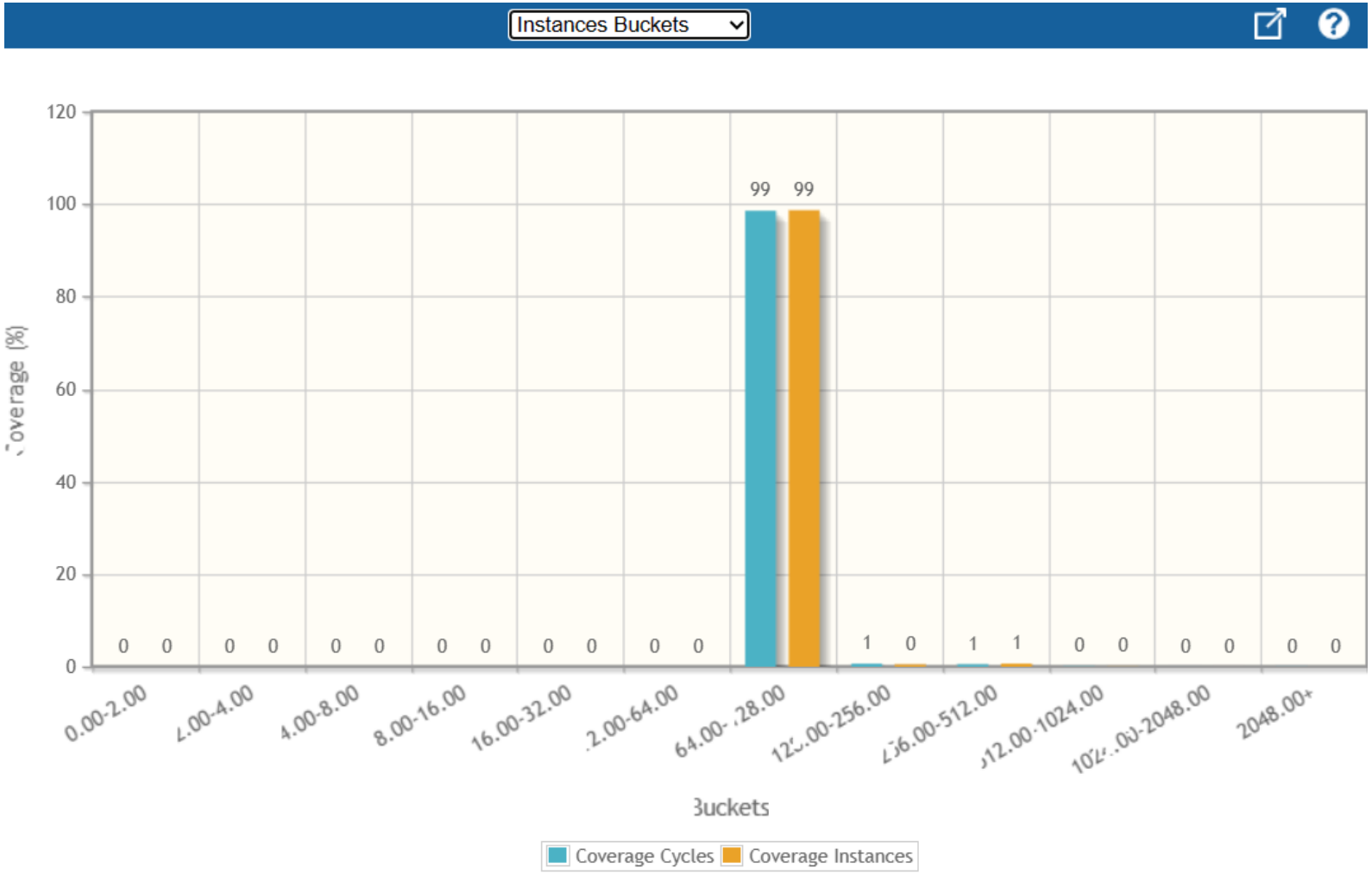
- Define Cycle per Iteration (CPI) and not execution time as the metric of interest.
- Define CPI Bucket: Bucket1: CPI between 1 and 2 cycles, Bucket2: CPI between 2 and 4 cycles, Bucket3: CPI between 4 and 8 cycles and so on…

REMARK: lower and upper bound of a CPI bucket is a parameter and as such can be changed.

- Perform a full tracing measurement of all instances for a few selected loops. All individual measurements are not kept:
  - Compute general metrics: min, max, average values across all instances
  - Count number of instances per CPI bucket: for example count the number instances belonging to Bucket1 i.e. instances with a CPI between 1 and 2 cycles
  - For each CPI bucket, keep 31 instance numbers which will be used in a more detailed performance analysis.

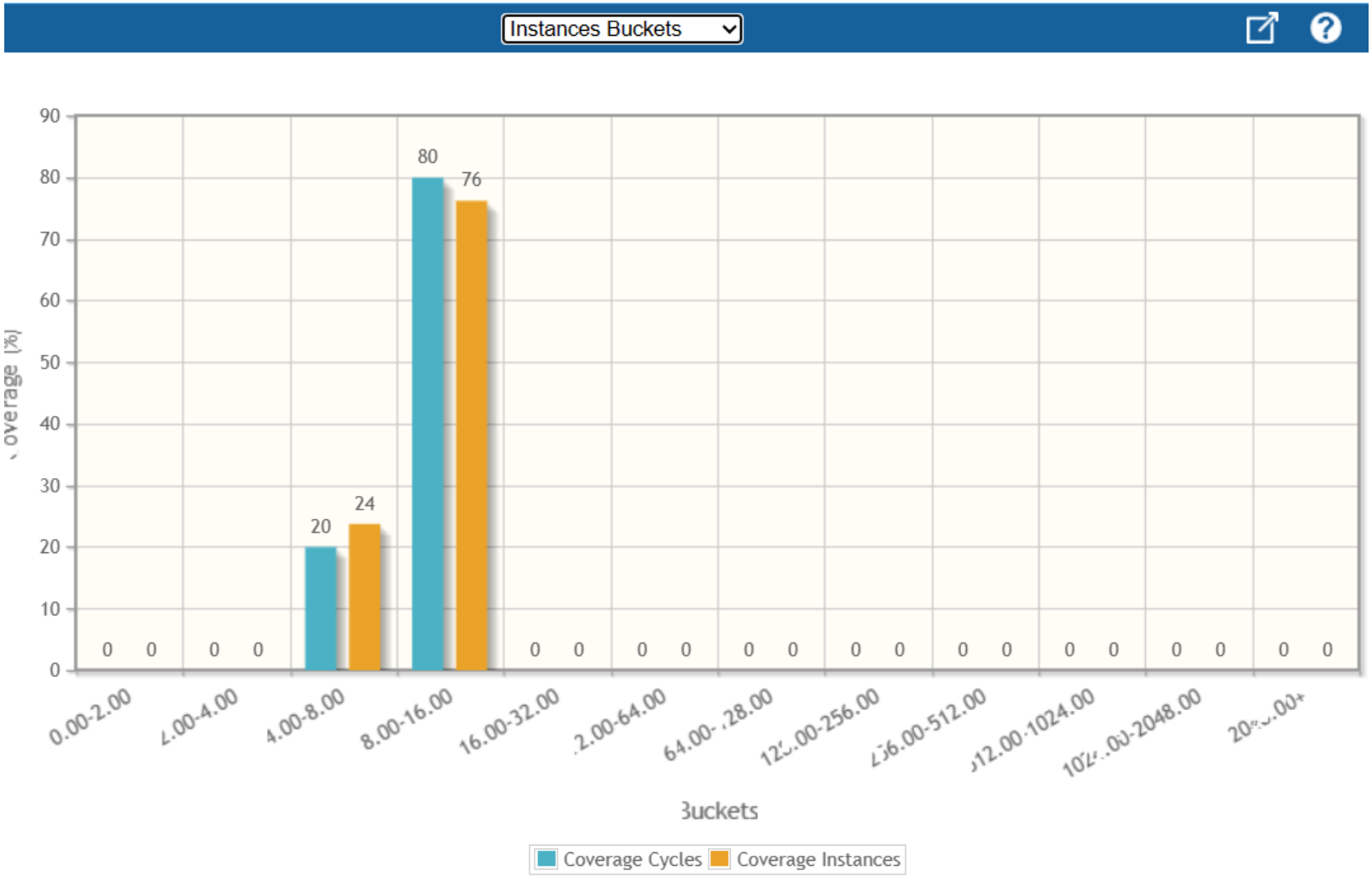
REMARK: value of 31 has been chosen so as to be able to build worthwhile statistics

CPI DISTRIBUTION: 1 BAR



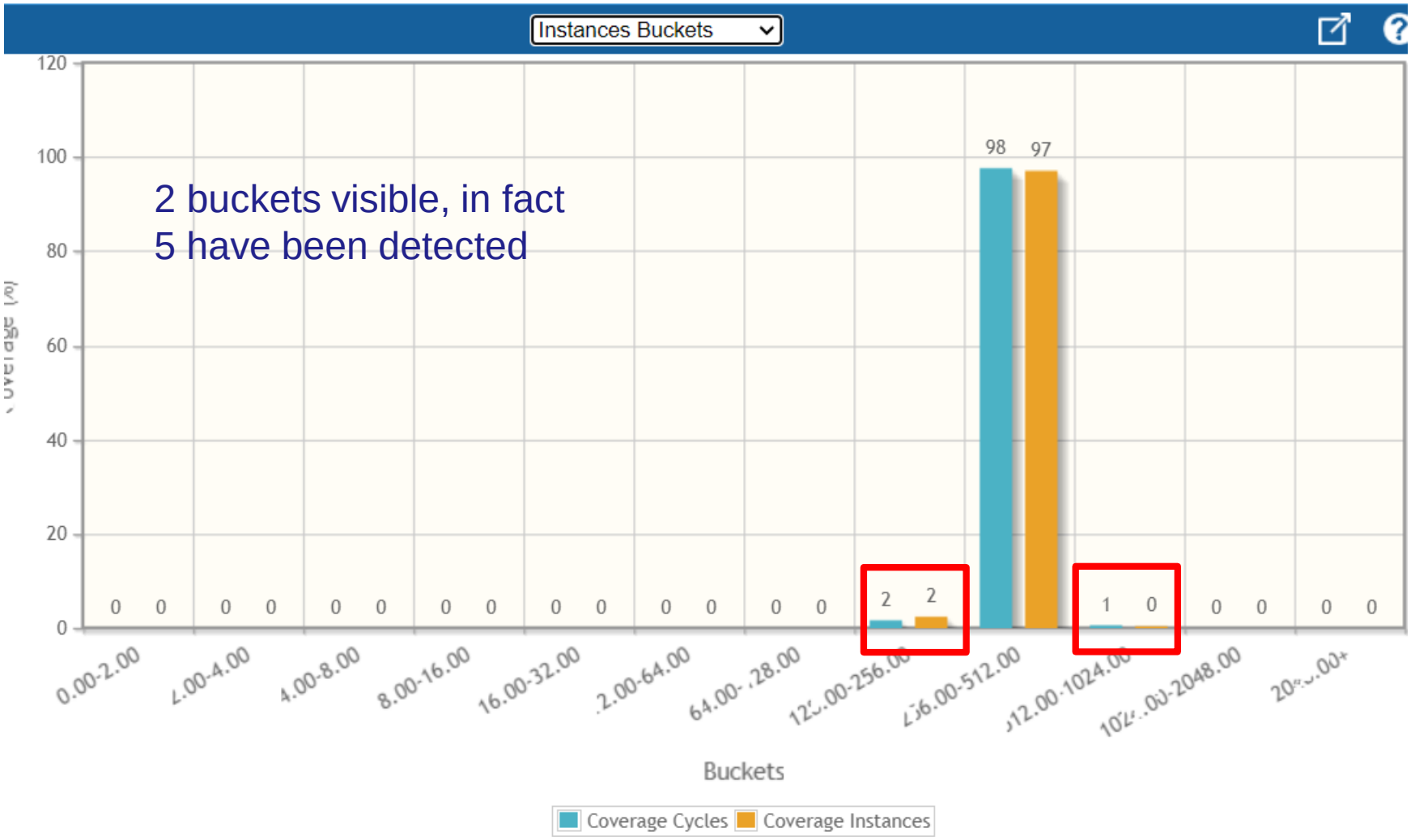
QMCPACK 2 threads running on SKL

CPI DISTRIBUTION: 2 BARS



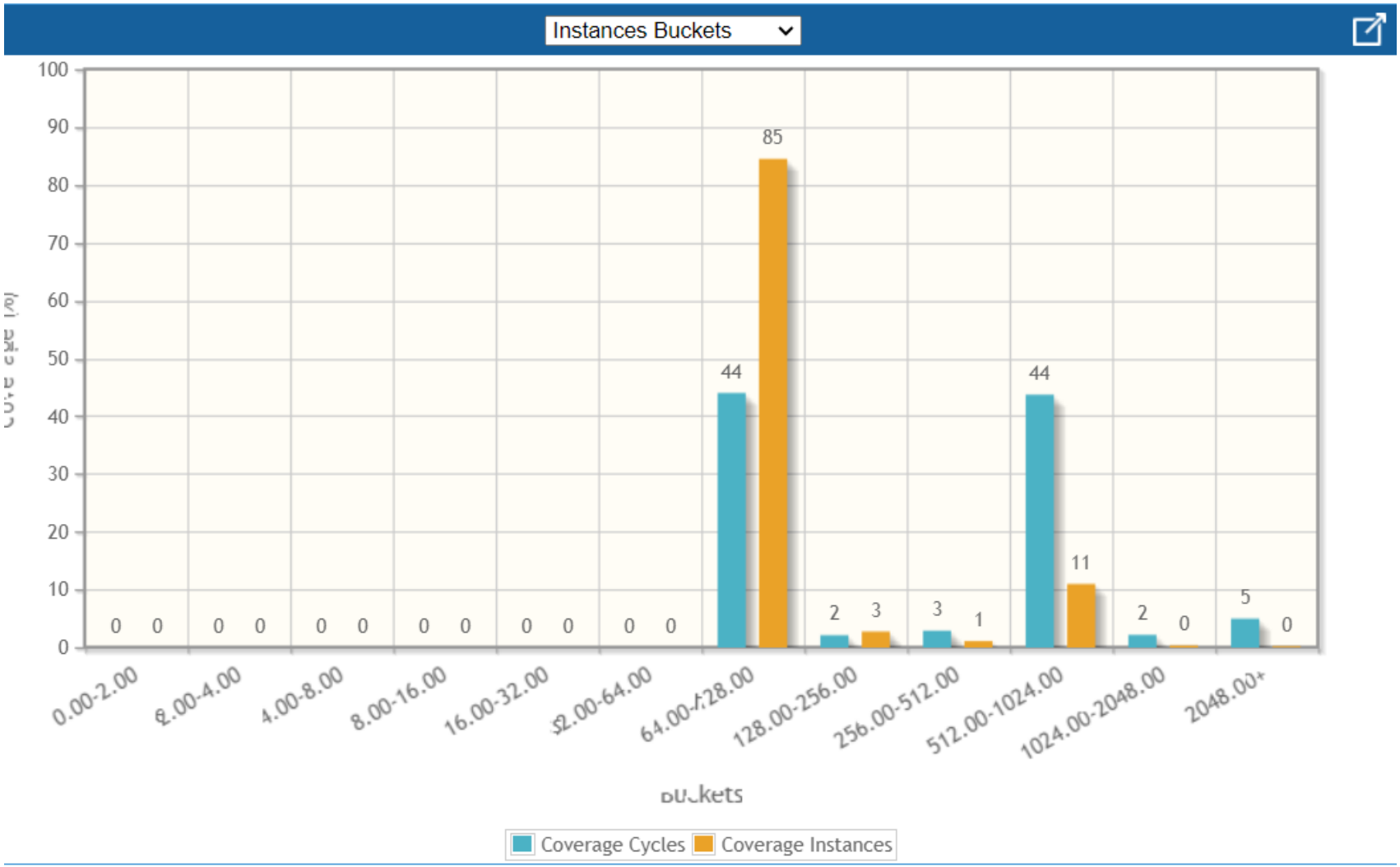
CHAMP (Twente Univ/TREX) 1 threads running on SKL

CPI DISTRIBUTION: 2 BARS



MINIQMC (proxy app QMCPACK) 16 threads running on SKL
Hottest loop

CPI DISTRIBUTION: MULTIPLE BARS



MINIQMC (proxy app QMCPACK) 16 threads running on SKL
Second hottest loop

GENERAL ORGANIZATION

- First Run: identify loop of interest
  - INPUT: whole program
  - OUTPUT: a few loop identifiers
  - By default, same loop identifiers for all threads
- Second Run: identify buckets
  - INPUT: A few loop identifiers
  - OUTPUT: for each loop, a few buckets and for each bucket, a few instance numbers
  - Buckets (and instances within a bucket) are generated independently for each thread
- Third Run: generate performance numbers using DECAN
  - Measurements are performed per thread and per bucket
  - For each loop and bucket in a thread, a typical representative will be the median

Buckets / Instances

		Thread 1	Thread 2	Thread 3
Loop		Bucket 2	Bucket 2	Bucket 2
	Instances	8, **12**, 15, **35**, 36, …	7, 12, **13**, 16, **17**, …	1,5, 8, **9**, …
	Median	**35**	**17**	
		Bucket 3	Bucket 3	
		1, 3, 5, **6**, 13, **14**, 18	4, **5**, 6, …	
	Median	**14**		

Instances in red are those that will be monitored by tracing

MAQAO DECAN: Decremental Analysis

- Goal: modify the application to
  - Identify cause of bottlenecks
  - Estimate associated performance impact

- Differential analysis:
  - Targets innermost loops
  - Transforms loops
  - Compare performance of original and transformed copy

- Transformations
  - Remove or modify groups of instructions
  - Targets memory accesses or computation
  - Modified loops provide wrong results but that's OK because we are only interested in performance. All of the modified loops are run in a protected sand box environment.

MAQAO DECAN Transformations

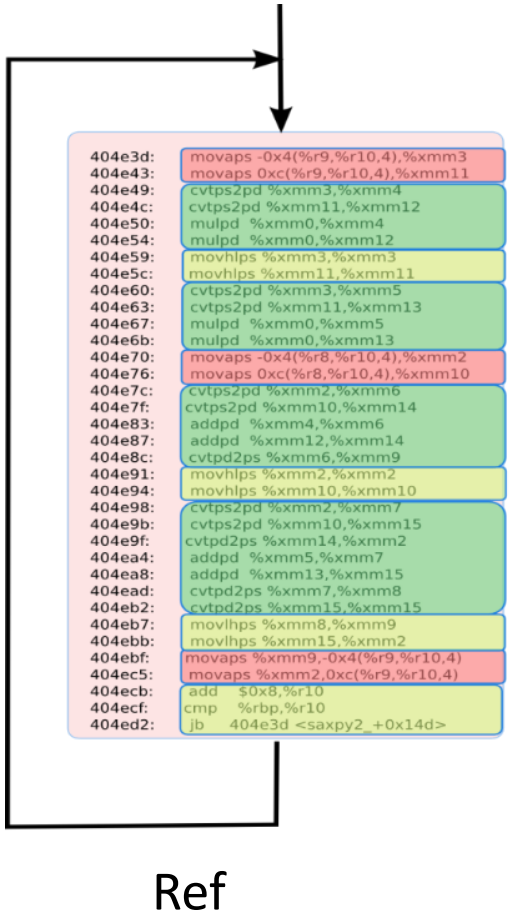
Typical transformations/variants:

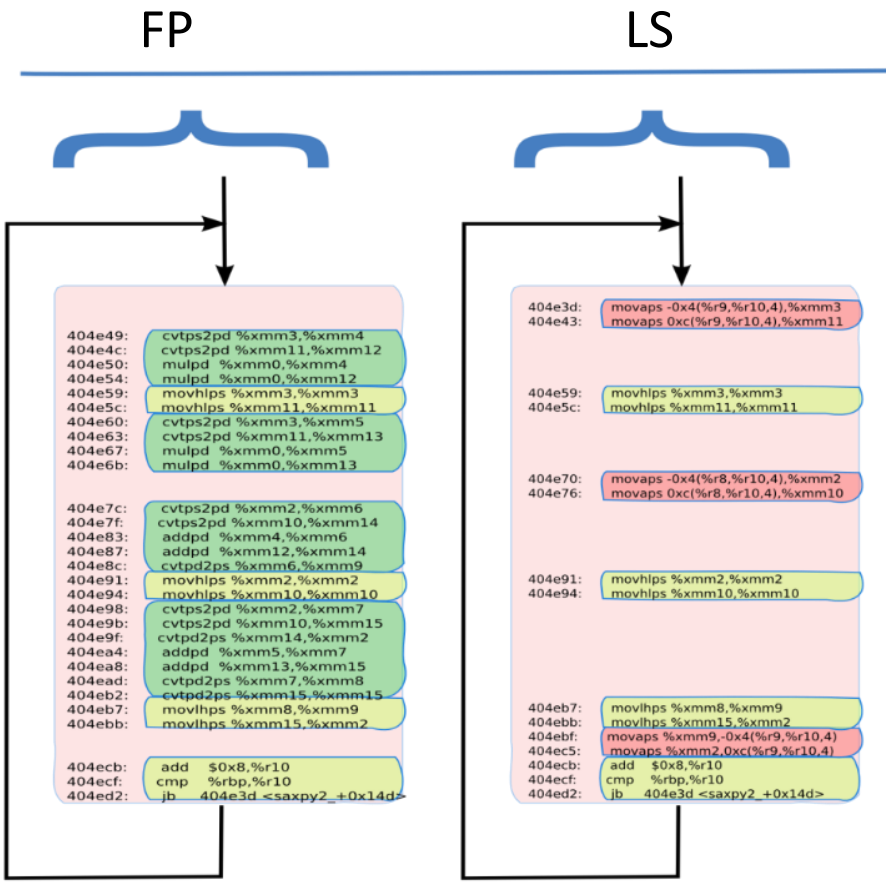
- **FP:** only FP arithmetic instructions are preserved
  - => loads and stores are removed
- **LS:** only loads and stores are preserved
  - => compute instructions are removed
- **FES:** only control flow instructions are preserved
  - => compute instructions are removed

Comparing T(FP) (Time spent in FP variant) and T(LS) (Time spent in LS variant) allows us to quantify how much a loop is CPU bound versus data access bound

- **DL1:** memory references replaced with constant memory address
  - => for loops, data now accessed from L1: precise impact of perfect blocking

MAQAO DECAN Example





STABILITY EVALUATION + ACROSS THREADS

For the 31 timing measurements performed within a bucket, Stability (STA) is defined as:

STA = (Median Time – Minimum Time) / Minimum Time

- Low values for STA means that there is little variation between instances
- On the contrary, large values for STA means large variations across instances

For each bucket/loop/thread, the median value within a bucket is used as a representative, then minimum, median and maximum can be computed over threads and STA across threads can be computed

Alternatively, we can bundle together for a given loop and a given bucket all of the measurements and again, minimum, median, maximum can be computed and another STA across threads is computed.

Expert Summary

▶ Colums Filter

ID	Coverage (% app. time)	ORIG (cycles per iteration)	STA (ORIG)	REF (cycles per iteration)	STA (REF)	FP (cycles per iteration)	STA (FP)	LS (cycles per iteration	STA (LS)	DL1 (cycles per teration)	STA (DL1)	FES (cycles per teration)	STA (FES)
▼ Loop 745	47.89	424.17	0.75	383.18	0.54	31.00	0.02	391.19	0.60	33.23	0.02	29.64	0.02
○ Bucket 9	97.71	424.17	0.75	383.18	0.54	31.00	0.02	391.19	0.60	33.23	0.02	29.64	0.02
○ Bucket 8	1.65	361.38	0.52	380.66	0.58	31.05	0.02	346.08	0.45	33.42	0.02	29.69	0.02
○ Bucket 10	0.575	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
○ Bucket 11	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
○ Bucket 12	0	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Hottest MINIQMC loop 16 threads run on SKLs

CYCLES PER ITERATION ACROSS THREADS

```
)o_omp.cpp: 259 - 259
-------------------------
-+)
:Value_OMPoffload.hpp: 100 - 10
:x + zs] + c[2] * coefs[index +
```

← Bucket 9 - 97.71% →

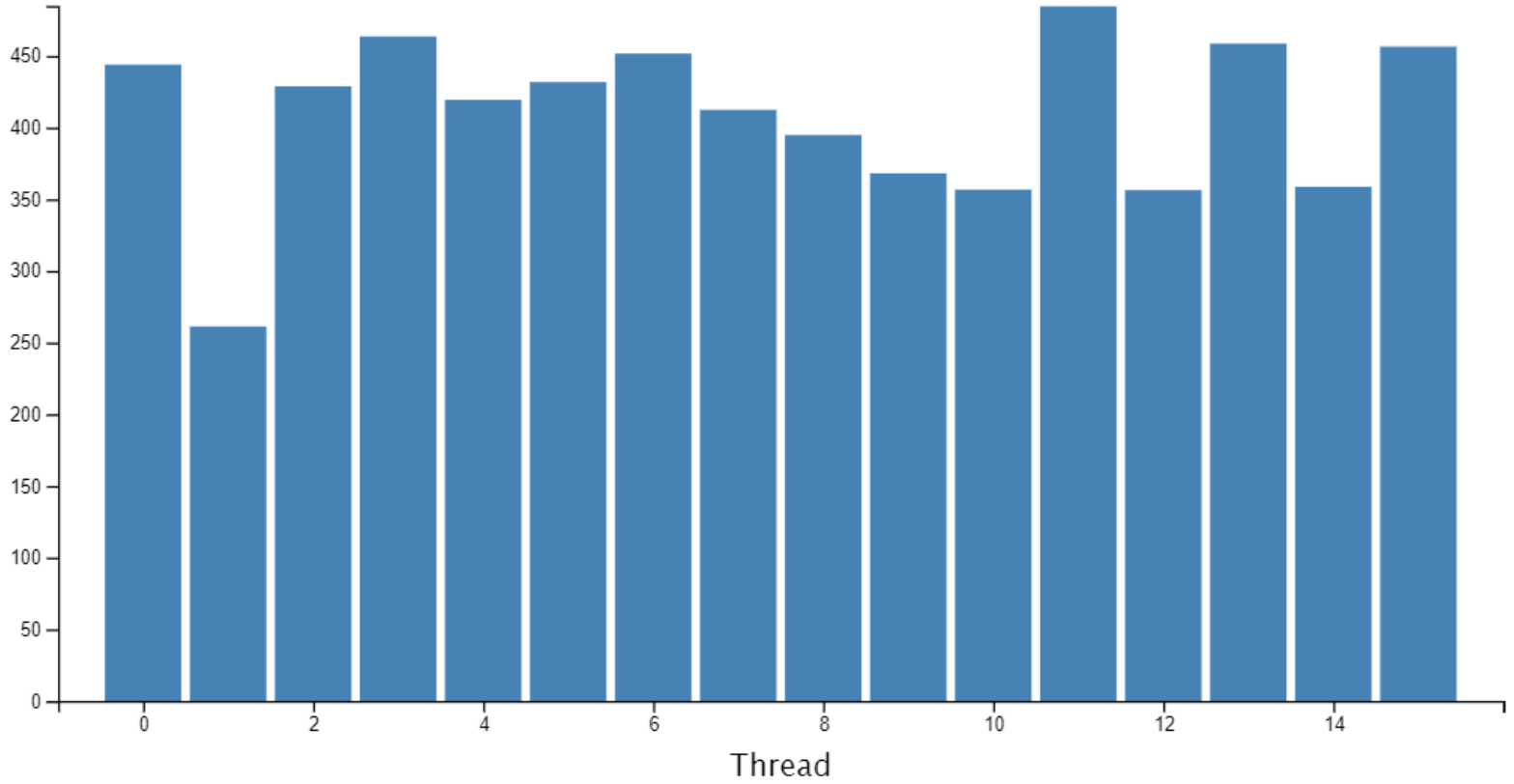
Show all results

Cycles Per Iteration (Median across each thread median values)

Metric (average per iteration except for Time and	ORIG	REF	DL1	LS	FP	FES
	81440.00	73571.00	6380.00	75109.00	5952.00	5691.00
CPI MIN	328.45	267.47	32.69	266.14	30.56	29.20
CPI MED	424.17	383.18	33.23	391.19	31.00	29.64
CPI AVG	406.90	372.33	33.62	374.73	31.34	29.69
CPI MAX	479.88	456.24	45.78	465.94	42.27	31.05

CPI ACROSS THREADS

CPI-ORIG



Deviation accross all processes: 36.152
Deviation accross all instances: 0.699

Score: 1 - average
Score: 2 - bad
Total Score: 3 (lower is better)

CYCLES PER ITERATION ACROSS THREADS (1)

DECAN

← Bucket 9 - 97.71% →

Show summarized results

Metric (average per iteration except for Time and Iteration Count)	ORIG						
	Min (Thread)	Med (Thread)	Avg (Thread)	Max (Thread)	Min (Instances)	Med (Instances)	Max (Instances)
Time	50170.00	81440.00	78576.75	93104.00	46616.00	79181.00	121238.00
CPI MED	261.30	424.17	409.25	484.92	242.79	412.40	631.45
Iteration Count	192.00	192.00	192.00	192.00	192.00	192.00	192.00

Min (Thread) means each thread is represented by its median across instances and Min is across these threads representatives
Min (Instances) means all of the measurements for threads and instances are merged together and Min is computed on that list

CYCLES PER ITERATION ACROSS THREADS (1)

LS						
Min (Thread)	Med (Thread)	Avg (Thread)	Max (Thread)	Min (Instances)	Med (Instances)	Max (Instances)
49226.00	75109.00	69750.25	82064.00	46982.00	73059.00	124000.00
256.39	391.19	363.28	427.42	244.70	380.52	645.83
192.00	192.00	192.00	192.00	192.00	192.00	192.00

FP						
Min (Thread)	Med (Thread)	Avg (Thread)	Max (Thread)	Min (Instances)	Med (Instances)	Max (Instances)
5940.00	5952.00	5950.88	5964.00	5846.00	5952.00	21780.00
30.94	31.00	30.99	31.06	30.45	31.00	113.44
192.00	192.00	192.00	192.00	192.00	192.00	192.00

CONCLUSION AND FUTURE DIRECTIONS

- Three important issues have been presented: unstabilities between runs, measurement quality, and performance variability
- We have demonstrated how MAQAO/OV tackles these 3 issues in a single environment and provides very useful insights;

FUTURE DIRECTIONS

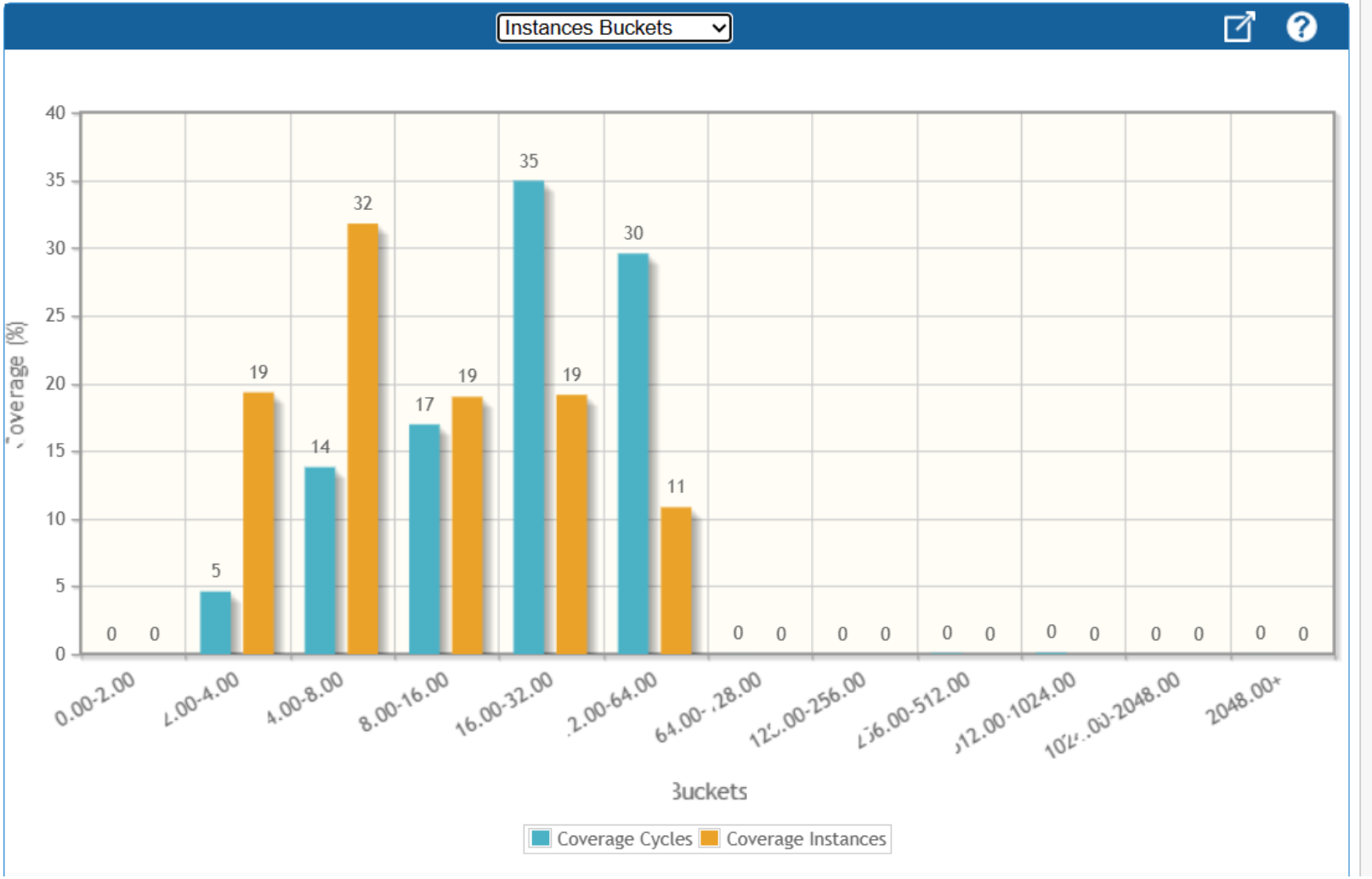
- Perform more advanced “statistics” on the data gathered
- For performance variability, MAQAO techniques are focussing on loops and they should be extended to parallel OpenMP regions and functions
- Define anomaly/unstability categories
- Automate anomaly/unstabilities detections
- Build a database of anomalies/unstability issues, including causes.
- Extend to multi-nodes

More on MAQAO

- MAQAO website: www.maqao.org
  - Documentation: www.maqao.org/documentation.html
    - Tutorials for ONE View, LProf and CQA
    - Lua API documentation
  - Latest release: http://www.maqao.org/downloads.html
    - Binary releases (2-3 per year)
    - Core sources
  - Publications around MAQAO: http://www.maqao.org/publications.html

BACKUP SLIDES

CPI DISTRIBUTION: MULTIPLE BARS



COMD 1 threads running on SKL

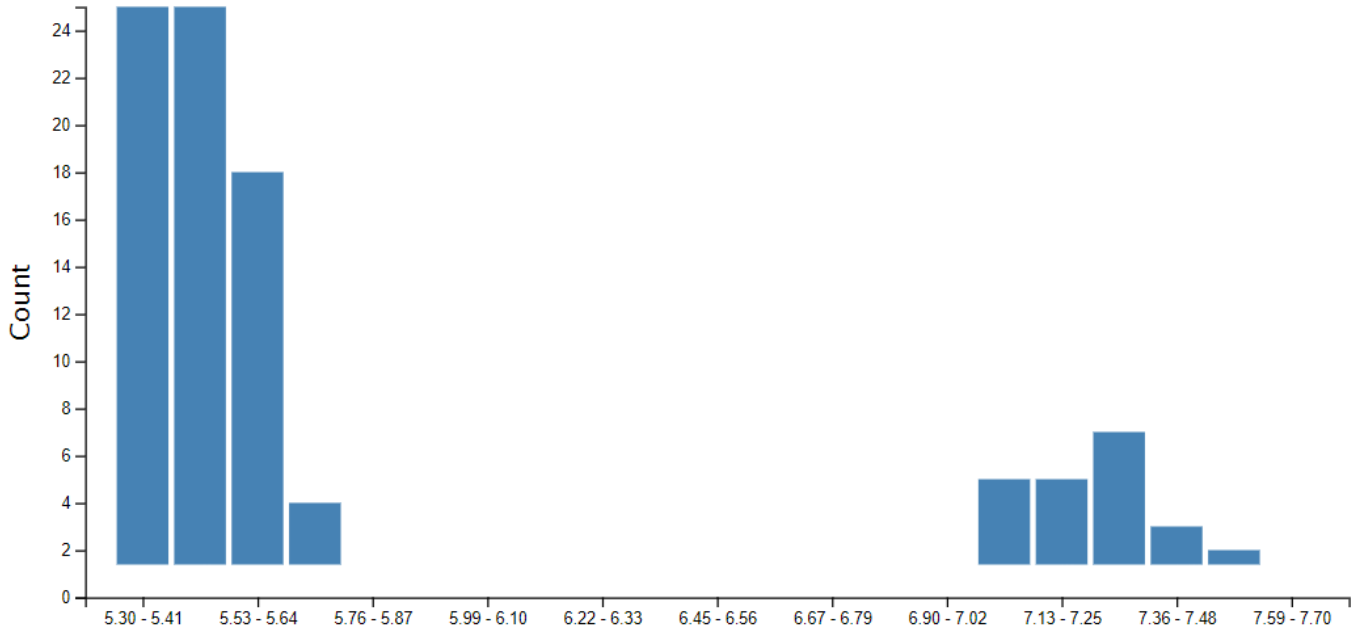
ISOBINARY: STABILITY RUNS (1)

Same code is run a large number of times to study measurement stability. Standard statistics (deciles) are computed. An extra xlsx file is produced.
KBL 4 cores. Lulesh2.0

Total Time

min	med	avg	max
5.3	5.53	5.93	7.59

Percentile Index	10	20	30	40	50	60	70	80	90	100
Value	5.37	5.4	5.43	5.47	5.52	5.57	5.67	7.03	7.26	7.59



ISOBINARY: STABILITY RUNS (2)

Same code is run a large number of times to study measurement stability.
Going at the function level allows to quickly identify delinquent functions
Min, max, avg, med are computed over the 100 runs.

Name	Module	min (Max Time Over Threads) (s)	avg (Max Time Over Threads) (s)	med (Max Time Over Threads) (s)	max (Max Time Over Threads) (s)
○ omp_get_num_procs	libgomp.so.1.0.0	0.71	1.14	0.9	2.24
○ std::vector >::operator[](unsigned long)	lulesh2.0	0.41	0.51	0.51	0.62
► CalcFBHourglassForceForElems(Domain&, double*, double*, double*, double*, double*, double*, double*, double, int, int) [clone ._omp_fn.7]	lulesh2.0	0.34	0.42	0.41	0.55
► EvalEOSForElems(Domain&, double*, int, int*, int) [clone ._omp_fn.17]	lulesh2.0	0.28	0.35	0.35	0.46
► CalcElemFBHourglassForce(double*, double*, double*, double (*) [4], double, double*, double*, double*)	lulesh2.0	0.19	0.25	0.24	0.33
► CalcEnergyForElems(double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double, double, double, double, double*, double*, double, double, int, int*) [clone ._omp_fn.20]	lulesh2.0	0.15	0.20	0.2	0.27
► CalcMonotonicQGradientsForElems(Domain&) [clone ._omp_fn.14]	lulesh2.0	0.16	0.20	0.2	0.26