Updates from the HEPiX Benchmarking Working Group

M. Alef (KIT), D. Giordano (CERN), M. Michelotto (INFN)

hepix-cpu-benchmark@hepix.org
https://twiki.cern.ch/twiki/bin/view/HEPIX/CpuBenchmark

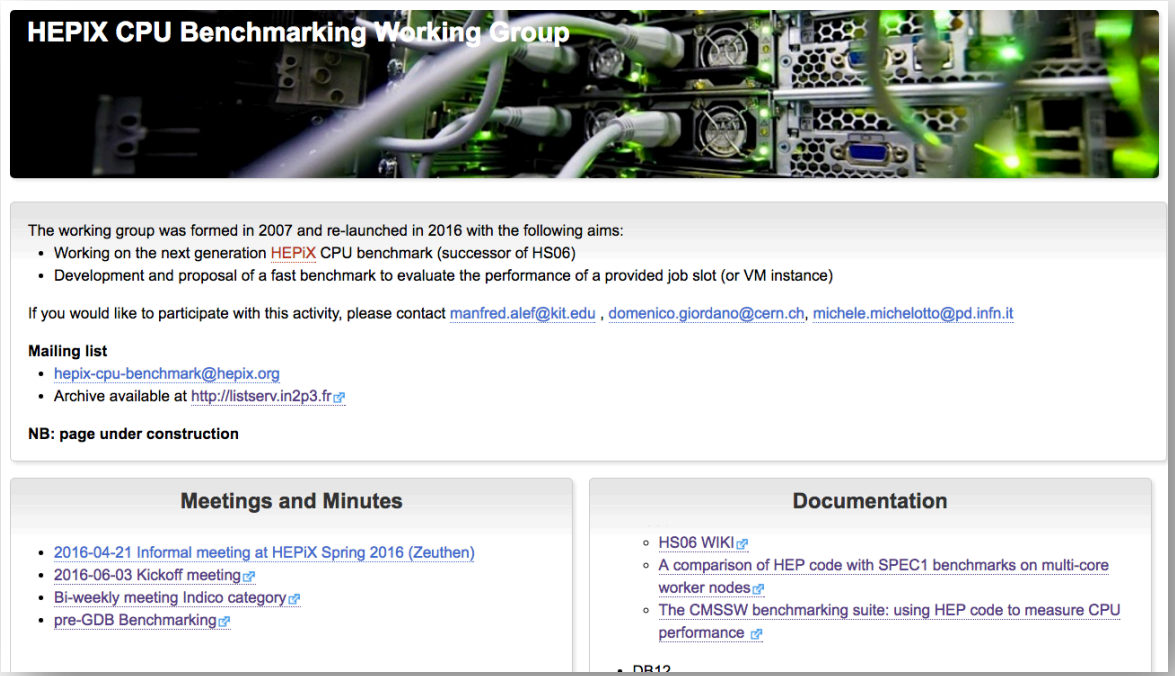
HEPiX Spring 2017 Workshop
26 April 2017

Mandate of the Working Group

- Investigate **scaling issues** between **HS06** and CPU intensive HEP workloads (i.e. EvtGen, Simulation)
- Study the **next generation of long-running benchmark** (successor of HS06)
- Evaluate **fast benchmarks**; identify their properties; provide recommendations to the community
  - To estimate the performance of a job slot (in traditional batch farms) or VM instances (in cloud environments)

Reference Twiki Page

- Track topics under study
  - Collect results
  - Document procedures to access dedicated resources and to run benchmarks
  - Includes link to previous HS06 Twiki page



HEPIX CPU Benchmarking Working Group

The working group was formed in 2007 and re-launched in 2016 with the following aims:

- Working on the next generation HEPiX CPU benchmark (successor of HS06)
- Development and proposal of a fast benchmark to evaluate the performance of a provided job slot (or VM instance)

If you would like to participate with this activity, please contact manfred.alef@kit.edu , domenico.giordano@cern.ch, michele.michelotto@pd.infn.it

Mailing list

- hepix-cpu-benchmark@hepix.org
- Archive available at http://listserv.in2p3.fr

NB: page under construction

Meetings and Minutes

- 2016-04-21 Informal meeting at HEPiX Spring 2016 (Zeuthen)
- 2016-06-03 Kickoff meeting
- Bi-weekly meeting Indico category
- pre-GDB Benchmarking

Documentation

- HS06 WIKI
- A comparison of HEP code with SPEC1 benchmarks on multi-core worker nodes
- The CMSSW benchmarking suite: using HEP code to measure CPU performance

https://twiki.cern.ch/twiki/bin/view/HEPIX/CpuBenchmark

Checkpoints

- Two main checkpoints until now
  - Feb. pre-GDB and Apr. GDB
  - Debates around the scenarios for fast benchmarks adoption
    - 😉 Forecasting job slot duration or checking performance of VM in cloud environments (large consensus)
    - 😕 Replacement of HS06 for site pledge and procurement
      - Large divergence of opinions
      - Limited instruction mix ➔ exposed to microarchitecture changes/optimization
      - Risk of missing all implications of that choice on the medium-long term
- Major effort to study in detail the fast benchmarks

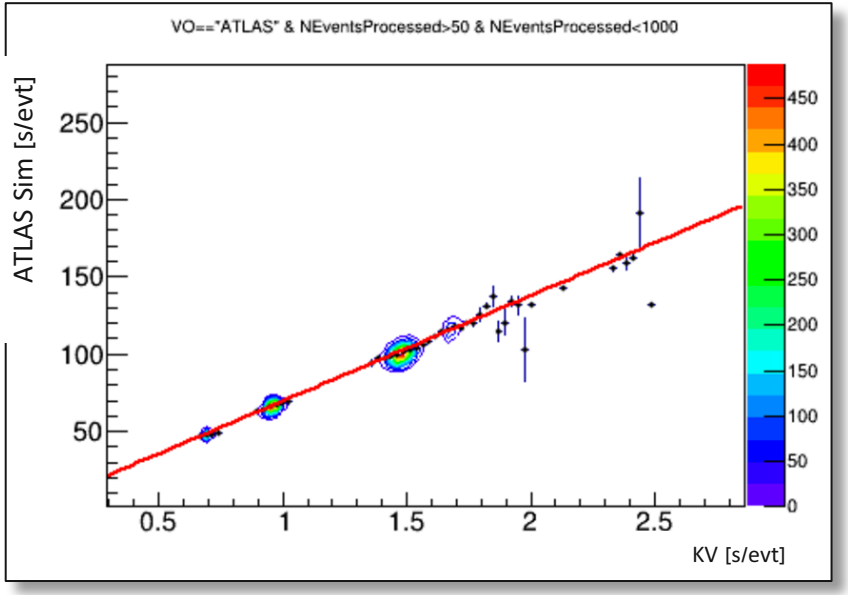
Fast Benchmarks

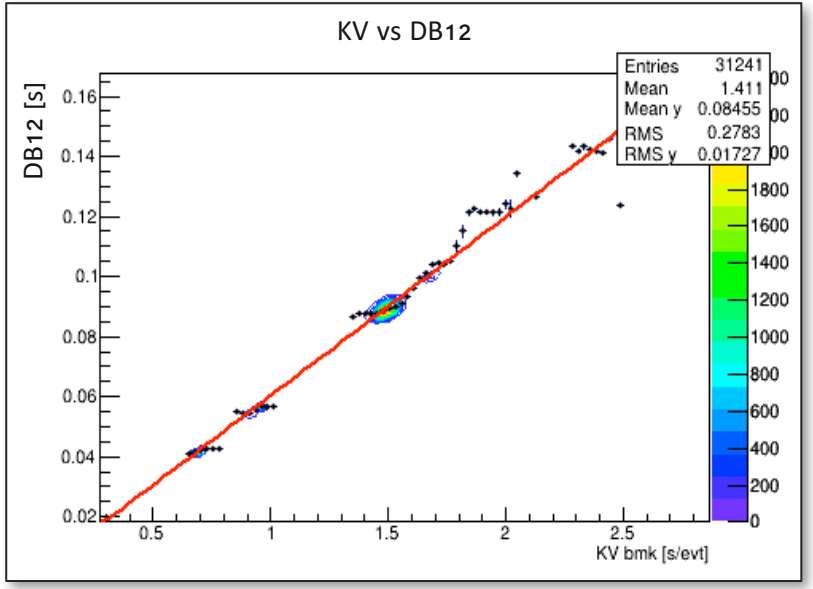
Started with 5 candidates

- ATLAS KV (KitValidation)
  - Mainly GEANT4. Default workload: 100 single muon event simulation
- DIRAC Benchmark 2012 (DB12), a.k.a. FastBmk, LHCbMarks
  - Python script: random.normalvariate() [see next slides]
- ROOT Stress test
- Legacy benchmarks: Whetstone, Dhrystone

- Systematic studies converge towards DB12 and Atlas KV
  - In particular DB12 seems to reproduce the job performance in Intel Haswell CPUs (a.k.a. “magic boost”)

KV performance Vs ATLAS Sim job





Azure A3 and D3 series

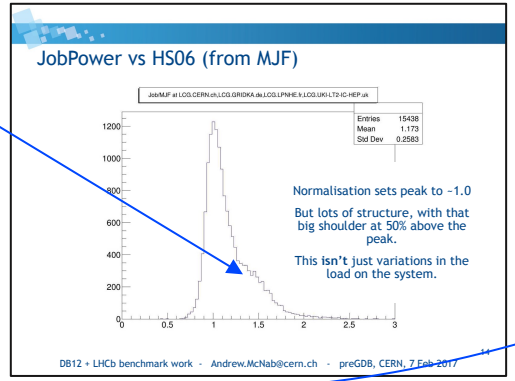
Running benchmarks and jobs in Commercial Clouds (VM)

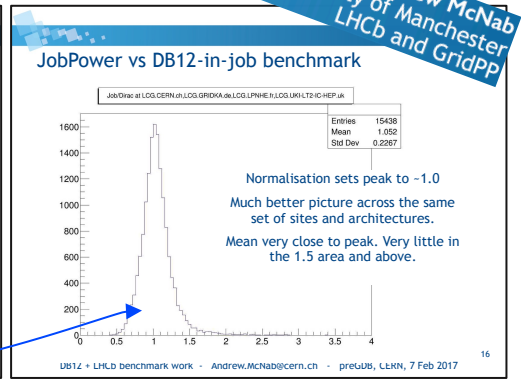
ATLAS Sim jobs Vs KV and DB12 Vs KV:

- At first order good linearity proven across different VM (and CPU) models

DB12 Vs jobs (LHCb and ALICE)

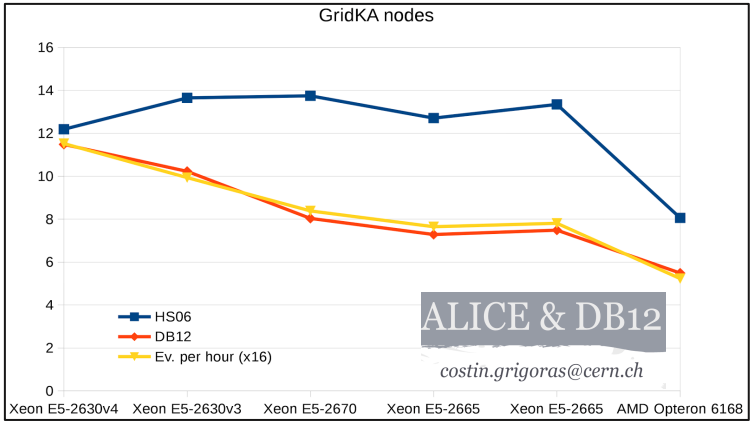
- LHCb
  - HS06 not a good predictor for MC
    - In particular for Intel Haswell CPUs
    - Not as bad for LHCb reconstruction jobs though
  - DB12 is much better





Andrew McNab
University of Manchester
LHCb and GridPP

- ALICE
  - Very good correlation of DB12 Vs MC
  - Large discrepancy respect to HS06



- NB: the average load of the server (i.e. number of running job slots) plays a major role in those evaluations (see next slides)

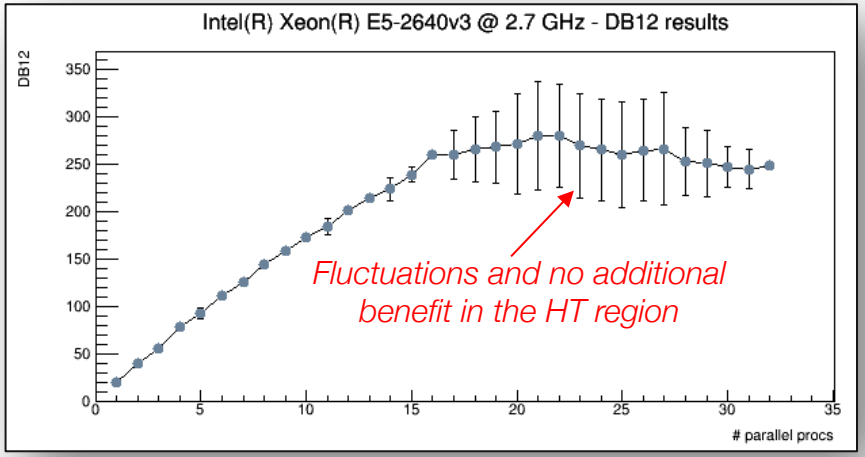
CMS Tests

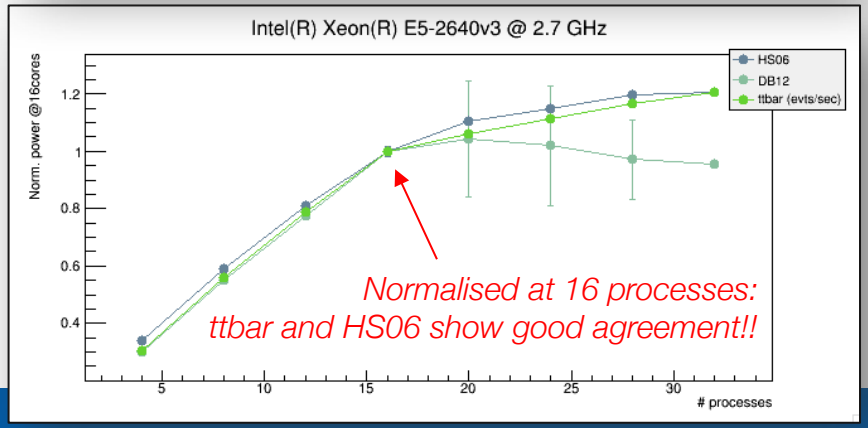
J. Flix https://indico.cern.ch/event/624813/

Preliminary comparison of DB12, KV, HS06 Vs CMS jobs

- Several dedicated nodes targeted for benchmarking
  - Sandy Bridge, Ivy Bridge, Haswell, AMD, ARM
  - ttbar simulation

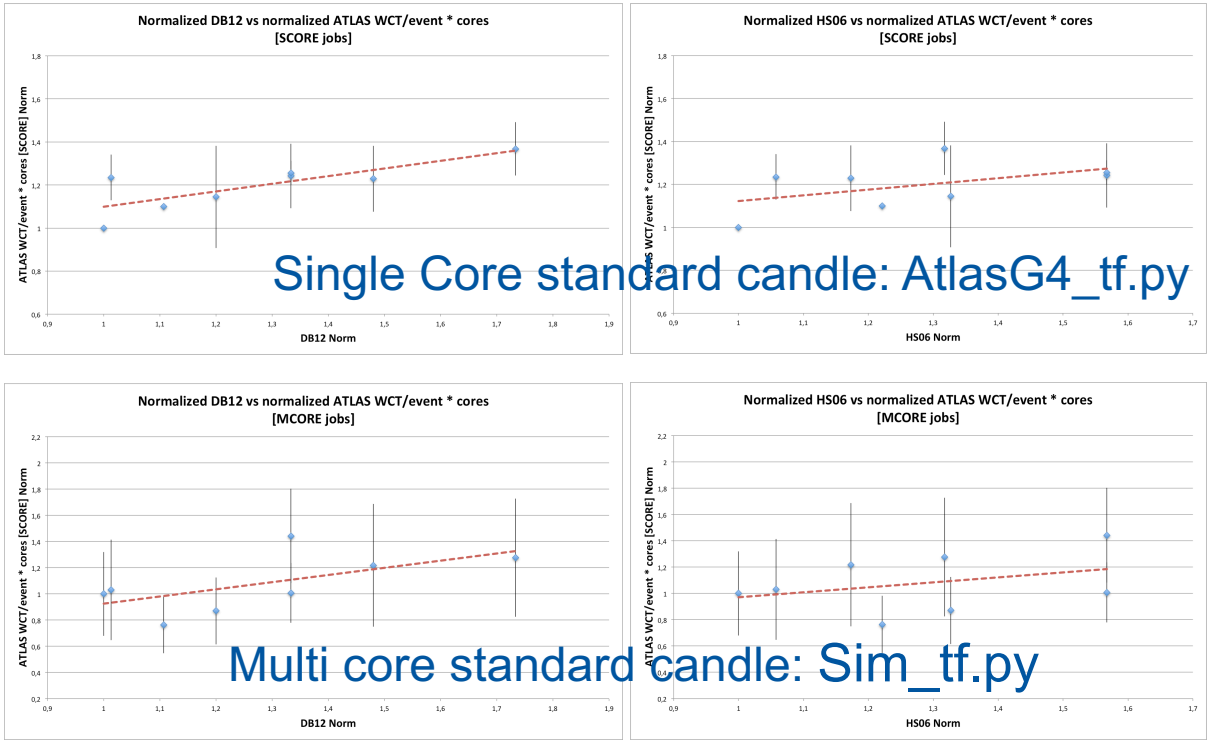
- Ongoing activity
  - Pilots instrumented to run fast benchmarks and collect results Vs any kind of job
  - CPU model reported in the job dashboards
    - This will enable also the passive benchmark approach (see next slides)





ATLAS

Alessandro De Salvo
on behalf of the ATLAS Distributed Computing group



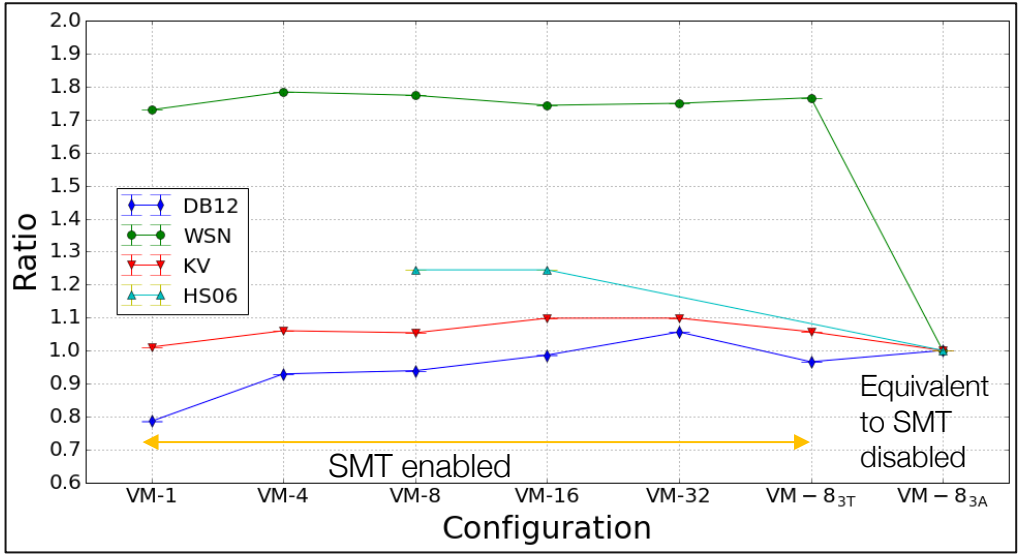
Single Core standard candle: AtlasG4_tf.py

Multi core standard candle: Sim_tf.py

- HammerCloud reference jobs (single and multi core) running on benchmarked resources at GridKa
- Further investigation is needed
  - In particular for multi core jobs

Scaling factors across VM configurations

- Benchmarking of Haswell servers (E5-2630 v3 CPUs) in virtual environment
- Partition the available compute resources in a number of VMs of same size
  - => Fully load the servers as done for HS06
- Study the ratio of performance among different configurations
  - VM sizes and SMT enabled/disabled
- The +20% gain in performance seen in HS06 with SMT=ON is not reproduced by DB12 and KV
  - Bigger discrepancy from DB12



DB12 studies

Seen the multiple and somehow diverging results on DB12 the working group decided to invest effort in additional studies

- Application profiling
- Reproducibility under different Python versions
- Effect of different implementations: C++ or using Numpy

DB12 (python) Features

```python
# Set up all the subprocesses
for i in range( copies ):
  results.append( multiprocessing.Value('d', 0.0) )
  processes.append( multiprocessing.Process( target = singleDiracBen

# Start them all off at the same time
for p in processes:
  p.start()
```

```python
# Possibly do one extra iteration to avoid tail effects when copies run in
for i in range( iterations + 1 + (1 if extraIteration else 0)):
  if i == 1:
    start = os.times()

  # Now the iterations
  for _j in xrange( n ):
    t = random.normalvariate( 10, 1 )
    m += t
    m2 += t * t
    p += t
    p2 += t * t

  if i == iterations:
    end = os.times()

cput = sum( end[:4] ) - sum( start[:4] )
```

```python
# Return DIRAC-compatible values
return { 'CPU' : cput, 'WALL' : wall, 'NORM' : calib * iterations / cput, 'UNIT' : 'DB12' }
```

Main components of the python script

- Multi – Processes spawn by `multiprocessing` module
- Per process:
  - Loop around random number generation
    - `random.normalvariate(10,1)` from `random` module
  - Eval CPU time (cput)
  - Get DB12 score as calib/cput
- Compute average, median, sum of DB12 individual scores

<u>Assumption</u>: DB12 well represents MC jobs because mainly dominated by random number generation, as are MC jobs

- Is this assumption correct? (see next slides)

https://github.com/DIRACGrid/DB12/blob/master/DIRACbenchmark.py

The Branch Prediction due to CPython Module

DB 12 - Functions profile

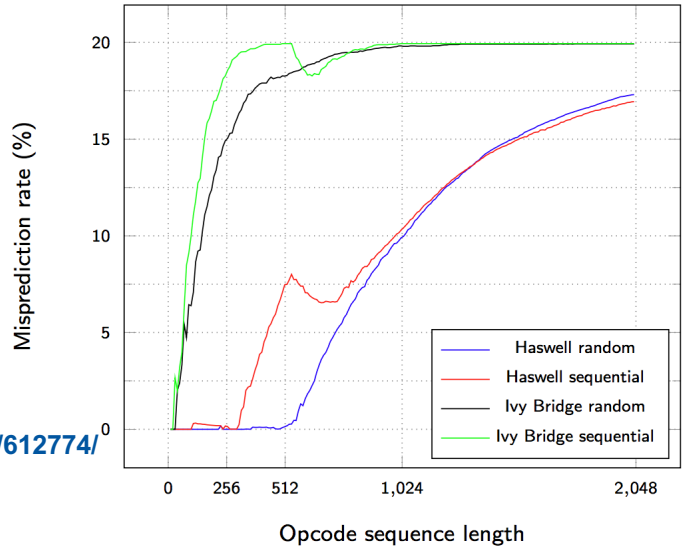
```
# Samples: 141K of event 'cycles:pp'
# Overhead  Command  Shared Object        Symbol
    41.50%  python   libpython2.7.so.1.0  [.] PyEval_EvalFrameEx
     6.44%  python   libpython2.7.so.1.0  [.] binary_op1
     4.72%  python   libm-2.17.so         [.] __ieee754_log_avx
     2.69%  python   libpython2.7.so.1.0  [.] float_mul
     2.29%  python   libpython2.7.so.1.0  [.] lookdict_string
     2.23%  python   libpython2.7.so.1.0  [.] float_div
     1.73%  python   _randommodule.so     [.] genrand_int32
     1.70%  python   libpython2.7.so.1.0  [.] PyFloat_FromDouble
     1.68%  python
```

Functions profile o

```
# Samples: 94K of e
# Overhead  Command
    37.60%  python
     7.48%  python
     6.65%  python
     3.68%  python
     2.27%  python
     2.13%  python
     2.11%  python
     2.09%  python
     1.94%  python
```

Functions profile o

DB 12 - Branch prediction perf



https://indico.cern.ch/event/612774/

- perf studies from M. Guerri have shown that the main contributor to DB12 is **`PyEval_EvalFrameEx`**
- Starting from Haswell models, this module benefits from a better branch prediction that boosts the DB12 performance

[see L. Atzori's talk]

Other Implementations of DB12

- DB12 Optimised Python version with Numpy (V. Innocente) : x10 faster than standard DB12
- DB12 C++ implementation (D. Giordano): x10 faster than standard DB12
  - Multi-processing using `fork`, `pipe`, same `normalvariate` algorithm
- Comparison of the **perf profile** for the three implementations shows that, contrary to the standard DB12 code, the Numpy and C++ versions are dominated by calls to math and rand libs

DB12 standard			DB12 numpy			DB12 C++		
Shared Object	Overhead [%]	Num of Symbols	Shared Object	Overhead [%]	Num of Symbols	Shared Object	Overhead [%]	Num of Symbols
libpython2.7.so.1.0	86.64	478	mtrand.so	37.82	36	DB12.exe	39.85	4
libm-2.17.so	5.19	4	libm-2.17.so	23.78	4	libm-2.17.so	39.75	3
_randommodule.so	4.06	63	libpython2.7.so.1.0	16.7	364	libc-2.17.so	19.8	3
math.so	2.63	73	umath.so	7.16	286	kernel.kallsyms]	0.48	12
kernel.kallsyms]	0.5	13	multiarray.so	5.62	318			
libpthread-2.17.so	0.44	3	libc-2.17.so	2.22	16			
libc-2.17.so	0.2	2	kernel.kallsyms]	0.97	43			
			libpthread-2.17.so	0.36	4			
Total	**99.66**			**94.63**			**99.88**	

- Documented @ http://cern.ch/go/Pq9T

DB12 Vs OS (and python) Versions

- DB12 scores are affected by changing python version
  - Variation of 10%-18%
- Ratio **DB12(32 cores) / DB12 (16 cores) = ~1**
  => no gain in SMT=ON
- As reference:
  - the scores of the C++ version
    are less affected by the different OS (~5%)
  - Ratio **DB12(32 cores) / DB12 (16 cores) = ~1.5**
    => 50% gain with SMT=ON
- **NB:** *This is **not** a suggestion to migrate DB12 to a C++ version, but just an example to highlight potential issues and discrepancies among implementations*
- Documented @ http://cern.ch/go/Pq9T

DB12 python		32 procs	
OS	**version**	**ratio_to_python2.6**	**ratio 32/16**
slc6-base	Python 2.6.6	1	1.00
CC7	Python 2.7.5	1.09	1.02
cc7-base	Python 2.7.5	1.09	1.04
python:2.7	Python 2.7.13	1.18	1.01
python:3	Python 3.6.0	1.05	0.98

DB12 C++		32 procs	
OS	**version**	**ratio_to_python2.6**	**ratio 32/16**
slc6-base	Python 2.6.6	1	1.49
CC7	Python 2.7.5	1.04	1.47
cc7-base	Python 2.7.5	1.03	1.46
(Debian 8.7) python:2.7	Python 2.7.13	1.05	1.49
(Debian 8.7) python:3	Python 3.6.0	1.05	1.48

Benchmark Comparison on Grid Nodes

Comparison on different HW models (M. Alef)

- Better scaling of DB12-cpp and DB12-np with HS06 than initial DB12 Python script
- The **+45%** boost appears **only** when running DB12 on

  # of slots =< # physical cores

  and goes down when SMT is enabled
  - Another effect of the lack of gain of DB12 with SMT enabled
  - NB: In SB also DB12 benefits of the 20% gain with SMT=ON

Double ratio respect to a given ref machine **(E5-2665 32 copies)** to the number of copies and to the HS06 of each machine (to remove frequency effect)

(bmk_X_CPU_Y_copies_N/bmk_HS06_CPU_Y_copies_N) / (bmk_X_CPU_REF_copies_M/bmk_HS06_CPU_REF_copies_M)

Hardware model		#copies	ratio	DB12/ HS06	DB12-cpp/ HS06	DB12-np /HS06
E5-2630v4	Broadwell	20	1	1.38	0.94	1.08
E5-2630v4		32	1.6	1.23	1.03	1.10
E5-2630v4		40	2	1.15	1.11	1.15
E5-2630v3		16	1	1.44	0.91	1.05
E5-2630v3		24	1.5	1.24	0.95	1.05
E5-2630v3	Haswell	32	2	1.08	1.05	1.07
E5-2660v3		20	1	1.48	0.94	1.05
E5-2660v3		32	1.6	1.24	1.04	1.10
E5-2660v3		40	2	1.17	1.12	1.14
E5-2665	Sandy B.	16	1	0.99	0.71	0.79
E5-2665		24	1.5	0.94	0.88	0.91
E5-2665		**32**	**2**	**1.00**	**1.00**	**1.00**

Dissecting Benchmarks (I)

M. Guerri

ITTF: https://indico.cern.ch/event/612774/
Internal Note http://cds.cern.ch/record/2257973?ln=en

Open Question

DB12 does not profit from SMT on Haswell/Broadwell architecture. KV apparently does the same. What is the reason behind this slowdown? Is it the same for both benchmarks? e.g. (hypothesis, not validated yet):

- DB12 is heavy on branches/jump. On Haswell, there are two execution ports that can execute branch instruction (compared to only one on IvyBridge. These are not ports exclusively for branch instructions). Validation of speculative fetch happen very fast keeping the pipeline always very busy.
- KV heavily profits from iTLB. When running two hardware threads, iTLB entries are thread specific. High number of conflicts.

17th February 2017 Benchmarking Working Group 1

ATLAS KV - Functions profile

```
# Samples: 125K of event ' cycles:pp', Event count (approx.): 99035639955
12.72%  athena.py  libm-2.17.so       [.] __ieee754_log_avx
 7.36%  athena.py  libm-2.17.so       [.] __ieee754_exp_avx
 5.51%  athena.py  libG4global.so     [.] G4PhysicsVector::Value
 3.64%  athena.py  libG4processes.so  [.] G4UniversalFluctuation::SampleFluctuations
 3.25%  athena.py  libm-2.17.so       [.] __GI___exp
 2.67%  athena.py  libm-2.17.so       [.] __sin_avx
 2.54%  athena.py  libMagFieldServices.so [.] BFieldCache::getB
 2.26%  athena.py  libG4tracking.so   [.] G4SteppingManager::DefinePhysicalStepLength
 2.13%  athena.py  libm-2.17.so       [.] __cos_avx
```

Functions profile on core 0 (single event)

```
# Samples: 141K of event ' cycles:pp', Event count (approx.): 111860905419
11.38%  athena.py  libm-2.17.so       [.] __ieee754_log_avx
 6.44%  athena.py  libm-2.17.so       [.] __ieee754_exp_avx
 4.90%  athena.py  libG4global.so     [.] G4PhysicsVector::Value
 3.38%  athena.py  libG4processes.so  [.] G4UniversalFluctuation::SampleFluctuations
 2.88%  athena.py  libm-2.17.so       [.] __GI___exp
 2.64%  athena.py  libm-2.17.so       [.] __sin_avx
 2.51%  athena.py  libMagFieldServices.so [.] BFieldCache::getB
 1.93%  athena.py  libG4tracking.so   [.] G4SteppingManager::DefinePhysicalStepLength
 1.83%  athena.py  libm-2.17.so       [.] __cos_avx
```

Functions profile on core 8 (single event)

March 17, 2017 Understanding applications performance 16

Unexpectedly large number of cycles spent at the beginning of each function (e.g. mov of literal value to register taking up 8%)
Hint of instruction cache misses?

NB: old Athena version 17.8.0.9 used in KV benchmark

Dissecting Benchmarks (II)

V. Innocente
https://indico.cern.ch/event/624828/contributions/2547881/attachments/1441812/2220330/Benchmarking.pdf

Conclusions 1

- Simulation (Geant4) (and python?) shows a high degree of code non-locality
  - Any different behavior in instruction pre-fetching will affect it much more than other benchmarks
- HS06 is memory greedy (at least compared to cms simulation& reconstruction)
  - Multiple instances running against common resources will scale worse than CMS Sim&Reco
  - It surely depends on the details of the cache hierarchy (arm, atom, knl, skylake, amd)

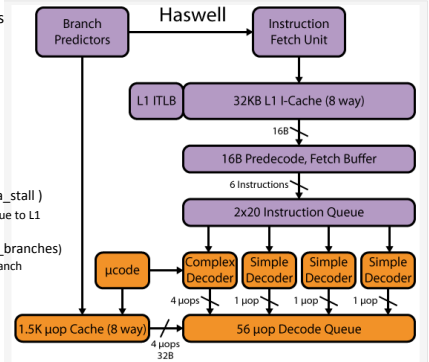
6/4/17 VI benchmark 9

Architecture: front end

Feeds "decoded" instructions to the scheduler

Affected by instruction non-locality (iCache-miss, iTLB misses) and misspredicted branches

Main metrics:
L1-icache-load-misses (icache.ifdata_stall)
Cycles where a code fetch is stalled due to L1 instruction cache miss.
branch-misses (br_misp_retired.all_branches)
This event counts all mispredicted branch instructions retired.



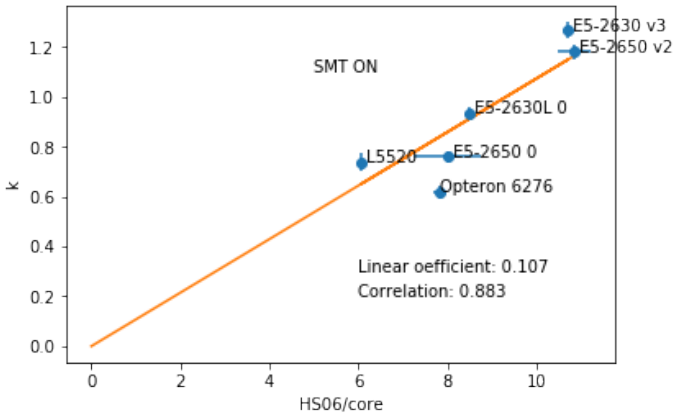
6/4/17 VI benchmark 4

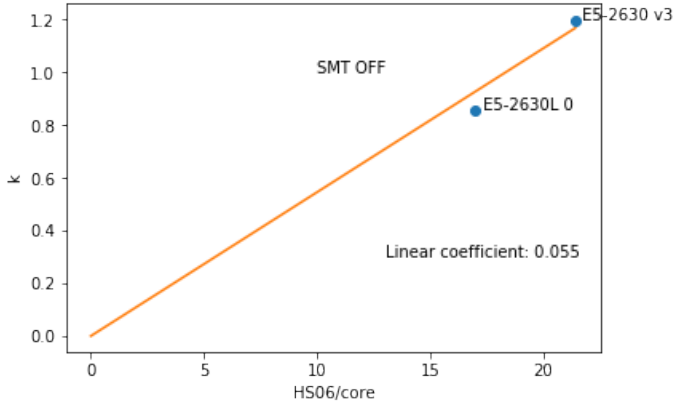
Passive Benchmarks of ATLAS Tier-0 CPUS

- Use real jobs to measure the relative speeds of different CPU models
  - Description of the analysis in the pre-GDB talk
- The analysis was applied to jobs run at the ATLAS Tier-0
  - Mainly reco jobs
  - Scaling generally good, two exceptions
    - Opteron way off
    - Haswell tends to perform better than what HS06 predicts: +10% with both SMT ON and OFF

[see A. Sciaba's talk, tomorrow]

A. Sciabà





Plan of Studies

- Start drafting requirements for the validation of the successor of HS06
  - Whatever it will be: HS17, HEP suite based on experiments' workloads, etc.
  - Aim: Scale, within a given accuracy, with a representative WLCG job mix
- Need to disentangle several effects
  - Bare-metal server Vs VMs, HT ON/OFF, different OS, -m32 Vs –m64, job variability, load on neighbor slots,…
- Identify testbed with representative set of hardware models
    - Include Non-default hardware models (e.g. ARM) of interest in HEP
- Perform reproducible studies
  - Document procedures
  - Experiments: identify and share representative job types (e.g. via CVMFS, containers, etc..)
  - Have access to monitoring data of production jobs

Summary of Intermediate Results

- DB12 (python version) has been (and still is) deeply studied
  - Scale well with ALICE and LHCb MC applications
  - BUT:
    - Runtime is dominated by libpython calls. Nothing to do with random number generation!
    - DB12 shows dependency from Python version, and it doesn't benefit from SMT enabled
    - +40% boost (respect to HS06) from Intel Sandy Bridge to Haswell only for ½ loaded servers (SMT enabled)
      - Discrepancy with HS06 goes down when processes running are x2 physical cores
      - The initial boost is due to a better branch prediction in the Haswell CPU frontend
  - ATLAS and CMS are instrumenting pilots to run, collect and study DB12 scores Vs production jobs (including multi-thread jobs)

- HS06
  - Preliminary study still shows good agreement among HS06 and CMS MC ttbar, when server fully loaded
  - Passive benchmarking
    - Discrepancies among HS06 and ATLAS reco jobs are within 10%

- Work in progress to setup a testbed for the HS06 successor