Profiling ATLAS CPU architectures

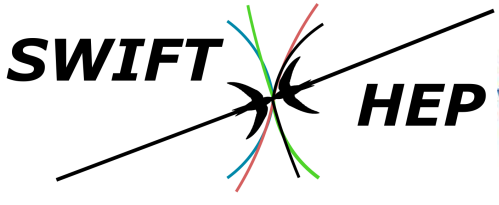
Jyoti Prakash Biswal

Rutherford Appleton Laboratory

GridPP49 & SWIFT-HEP05

The Cosener's House, Abingdon

29 March 2023



Instigation

- Since 2007, the CPUs on the grid have some baseline set of features.
- Anything newer is not used except via a special library, *e.g.*, **Intel Math Library function multi-versioning**.
- Therefore, the ability to run vectorised codes are passed over, not recognising the speed boost from the latest features.

  $\Rightarrow$ Familiarity with the CPU architectures on the grid should pave the way for more effective use of computing resources.

- **This study aims to gather specifics on CPU architectures via CPU flags using information from the jobs running on the grid.**

Status quo in ATLAS

- By and large, ATLAS codes are dominated by basic maths, C++ STL operations, and memory allocation/deallocation.
- These codes are compiled for this baseline because if compiled at a higher level and the CPU feature is unavailable, then athena will crash with "illegal instruction."
- The compiler can do auto-vectorisation on appropriate loops if the relevant architecture is enabled.
  - The baseline compiler can do some auto-vectorisation, but the more recent CPUs allow more parallel operations.
- Some codes will figure out at runtime what is available!
- In reality, there is no information on how much of the grid is which kind of CPU architecture – either in terms of which grid sites have them or how important they are.
- **There are potentially significant benefits that could be gained with compiler optimisation if they are allowed to use more vectorisation.**

CPU: features, instructions, and architectures

- CPU features:
  - Define a number of different processor attributes, *e.g.*, the presence of a floating-point unit (FPU).
  - Reflect on CPU operations at the current time; architecture dependent.
  - Description of CPU features: link-1; link-2.
- CPU instructions:
  - Patterns of bits, digits, or characters that correspond to machine commands.
  - The instruction set is specific to a class of processors using (mostly) the same architecture.
  - Successor or derivative processor designs often include instructions of a predecessor and may add new additional instructions.
- CPU architectures:
  - *System design*: physical computer system – all hardware parts of a computer.
  - *Instruction set architecture (ISA)*: the functions and capabilities of the CPU; what programming it can perform or process.
  - ***Microarchitecture***: computer organisation; defines the data processing and storage element and how they should be implemented into the ISA.
    - Four variations: ***x86-64-v1***, ***x86-64-v2***, ***x86-64-v3***, and ***x86-64-v4***.

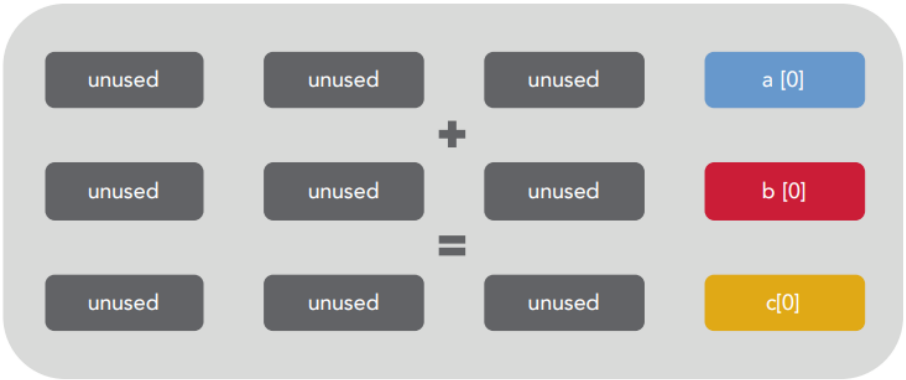
CPU: features, instructions, and architectures

- References: Wiki; Application Binary Interface (ABI).

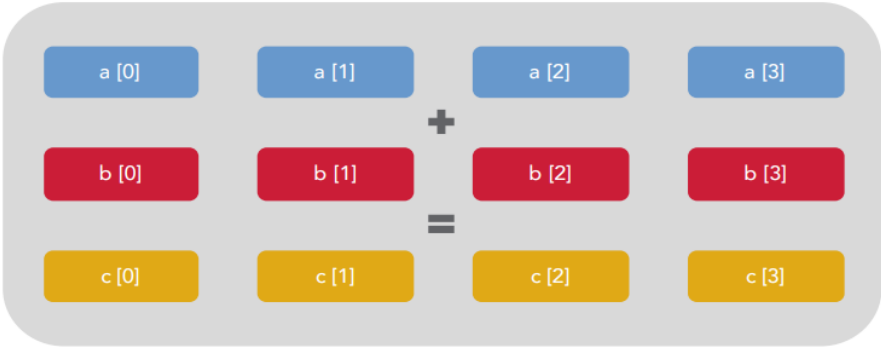
CPU microarchitecture levels		
Architecture	**Features**	**Example instructions**
x86-64-v1	CMOV CX8 FPU FXSR MMX OSFXSR SCE SSE SSE2	cmov cmpxchg8b fld fxsave emms fxsave syscall cvtss2si cvtpi2pd
x86-64-v2	CMPXCHG16B LAHF-SAHF POPCNT SSE3 SSE4_1 SSE4_2 SSSE3	cmpxchg16b lahf popcnt addsubpd blendpd pcmpestri phaddd
x86-64-v3	AVX AVX2 BMI1 BMI2 F16C FMA LZCNT MOVBE OSXSAVE	vzeroall vpermd andn bzhi vcvtph2ps vfmadd132pd lzcnt movbe xgetbv
x86-64-v4	AVX512F AVX512BW AVX512CD AVX512DQ AVX512VL	kmovw vdbpsadbw vplzcntd vpmullq _

A case in point: vectorisation

- A key tool to improve performance on modern CPUs.
- Converts an algorithm from operating on a single value at a time to operating on a set of values simultaneously.
- Modern CPUs provide direct support for vector operations where a **single instruction** is applied to **multiple data** (SIMD).
- **Advantages:**
  - A 512-bit CPU could hold 16 32-bit single precision doubles and do a single calculation.
    $\Rightarrow$ 16 times faster than executing a single instruction at a time.
    $\Rightarrow$ Combination with threading and multi-core CPUs leads to enormous performance gains.
  - The individual vector (array) elements are added in sequence in a serial calculation.



[Scalar mode: unused additional spaces in CPU.]



[Vector mode]

Vectorisation means optimising the algorithm to utilise SIMD instructions in the processors.

Vectorisation and CPU instructions/architectures

Instruction: SSE4 (*Streaming SIMD Extensions 4*)

- Architecture: x86-64-v2
- Processor: 128-bit
- Simultaneous operations on: **four** 32-bit single-precision floating point numbers / **two** 64-bit double-precision floating point numbers.

Instruction: AVX2 (*Advanced Vector Extensions 2*)

- Architecture: x86-64-v3
- Processor: 256-bit
- Simultaneous operations on: **eight** 32-bit single-precision floating point numbers / **four** 64-bit double-precision floating point numbers.

Instruction: AVX512 (*Advanced Vector Extensions 512*)

- Architecture: x86-64-v4
- Processor: 512-bit
- Simultaneous operations on: **sixteen** 32-bit single-precision floating point numbers / **eight** 64-bit double-precision floating point numbers.

AVX512 $\sim$ **2**$\times$**AVX2**; **AVX2** $\sim$ **2**$\times$**SSE4**.

Impact of newer CPU features (a recent study)

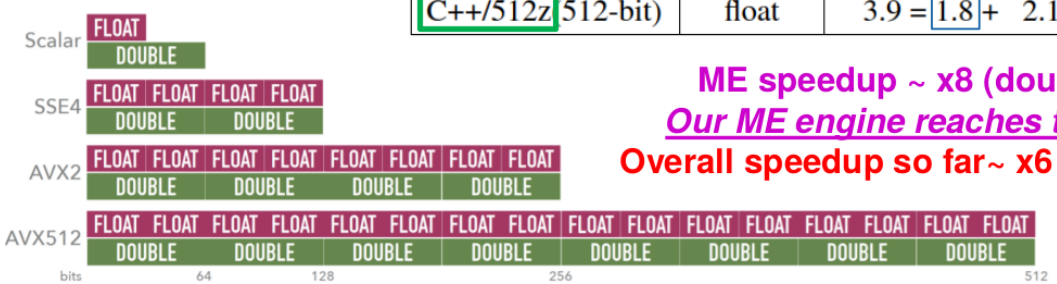
- **Speeding up Madgraph5_aMC@NLO through data parallelism: CPU vectorisation (A. Valassi's talk)**.
- On CPUs, in vectorised C++, the maximum **x8/x16** (double/float) SIMD speedup is reached for Matrix Elements (MEs) alone.
  - The speedups achieved for the overall workflow are slightly lower due to *Amdahl's law*, but not much.
  - *e.g.*, current overall speedup is **x6/x10** (double/float) for $gg \to t\bar{t}gg$ on one CPU core.

ACAT2022

$gg \to t\bar{t}gg$	MEs precision	$t_{TOT} = t_{Mad} + t_{MEs}$ [sec]	madevent N_{events}/t_{TOT} [events/sec]	madevent N_{events}/t_{MEs} [MEs/sec]	standalone
Fortran(scalar)	double	37.3 = 1.7 + 35.6	2.20E3 (=1.0)	2.30E3 (=1.0)	—
C++/none(scalar)	double	37.8 = 1.7 + 36.0	2.17E3 (x1.0)	2.28E3 (x1.0)	2.37E3
C++/sse4(128-bit)	double	19.4 = 1.7 + 17.8	4.22E3 (x1.9)	4.62E3 (x2.0)	4.75E3
C++/avx2(256-bit)	double	9.5 = 1.7 + 7.8	8.63E3 (x3.9)	1.05E4 (x4.6)	1.09E4
C++/512y(256-bit)	double	8.9 = 1.8 + 7.1	9.29E3 (x4.2)	1.16E4 (x5.0)	1.20E4
C++/512z(512-bit)	double	6.1 = 1.8 + 4.3	1.35E4 (x6.1)	1.91E4 (x8.3)	2.06E4
C++/none(scalar)	float	36.6 = 1.8 + 34.9	2.24E3 (x1.0)	2.35E3 (x1.0)	2.45E3
C++/sse4(128-bit)	float	10.6 = 1.7 + 8.9	7.76E3 (x3.6)	9.28E3 (x4.1)	9.21E3
C++/avx2(256-bit)	float	5.7 = 1.8 + 3.9	1.44E4 (x6.6)	2.09E4 (x9.1)	2.13E4
C++/512y(256-bit)	float	5.3 = 1.8 + 3.6	1.54E4 (x7.0)	2.30E4 (x10.0)	2.43E4
C++/512z(512-bit)	float	3.9 = 1.8 + 2.1	2.10E4 (x9.6)	3.92E4 (x17.1)	3.77E4

512y = AVX512, ymm registers
512z = AVX512, zmm registers

The latter is only better on nodes with 2 FMA units (here an Intel Gold 6148)



ME speedup ~ x8 (double) and x16 (float) over scalar Fortran
Our ME engine reaches the maximum theoretical SIMD speedup!
Overall speedup so far~ x6 (double) and x10 (float) over scalar Fortran (Amdahl's law)

Sources of information

- CPU flags: CPU features + CPU instructions.
  - Utilised in this study for making the decision on CPU architectures.
- The flags are obtained via –
  - /proc/cpuinfo →

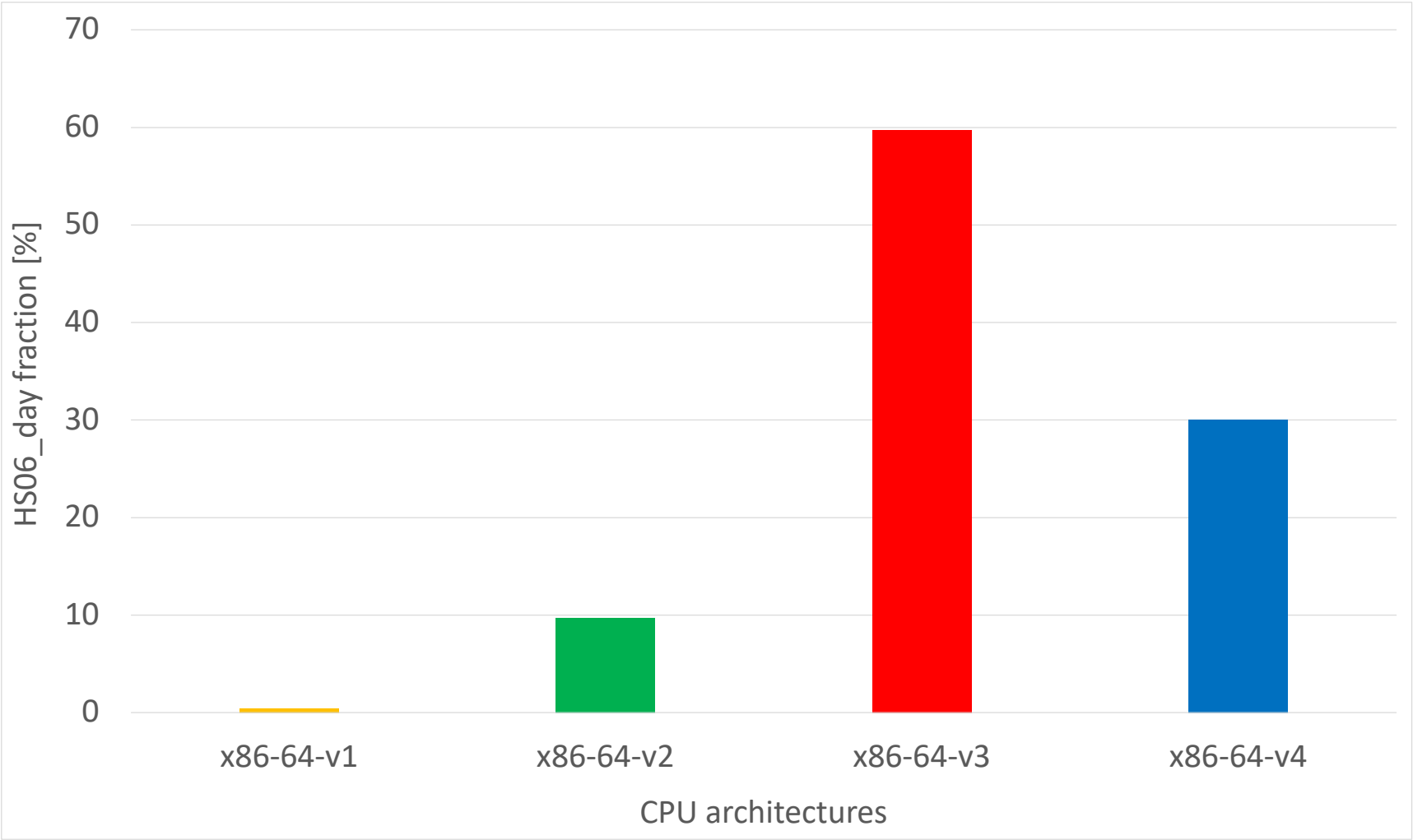
    e.g., **flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology eagerfpu pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd rsb_ctxsw ibrs ibpb stibp fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx smap xsaveopt arat md_clear spec_ctrl intel_stibp**
  - The same info is also stored in the **pilot logs**.
  - The pilot logs for all jobs are accessible via Big PanDA as well as apfmon Cloud.
    - There is a specific period through which the pilot logs are available on Big PanDA; the best approach is to download those.
- Also, kibana has quite some statistics available (*sans* CPU flags as of now).
- Historical data obtained from Big PanDA and kibana are made use of for this work.
- A Python script is designed for the purpose.

Procedure to decide on CPU architectures

- Deciding on the definition of CPU architectures based on CPU flags is not trivial.
- The following recommendations on CPU flags were considered before making a decision:
  - Naive/simplified:
    - SSE4_2.* → x86-64-v2.
    - AVX2.* → x86-64-v3.
    - AVX512.* → x86-64-v4.
  - GNU Compiler Collection (GCC): x86-64-v2, x86-64-v3, x86-64-v4.
- **The finalised one:**
  - Modified GCC lists *i.e.*, LAHF_SAHF ⟶ LAHF_LM; LZCNT ⟶ ABM; removal of SSE3.
    i.e.,
    - x86-64-v2 = [MMX, SSE, SSE2, LAHF_LM, POPCNT, SSE4_1, SSE4_2, SSSE3]
    - x86-64-v3 = [MMX, SSE, SSE2, LAHF_LM, POPCNT, SSE4_1, SSE4_2, SSSE3, AVX, AVX2, F16C, FMA, ABM, MOVBE, XSAVE]
    - x86-64-v4 = [MMX, SSE, SSE2, LAHF_LM, POPCNT, SSE4_1, SSE4_2, SSSE3, AVX, AVX2, F16C, FMA, ABM, MOVBE, XSAVE, AVX512F, AVX512BW, AVX512CD, AVX512DQ, AVX512VL]
- **AMD CPUs do not qualify for x86-64-v4 under the above criteria!**
- CPUs w/ ARM, High Performance Computing (HPC) are NOT examined.
- Presently, the results are the same as that of naive/simplified criteria.

CPU popularity architecture-wise (all grid sites)

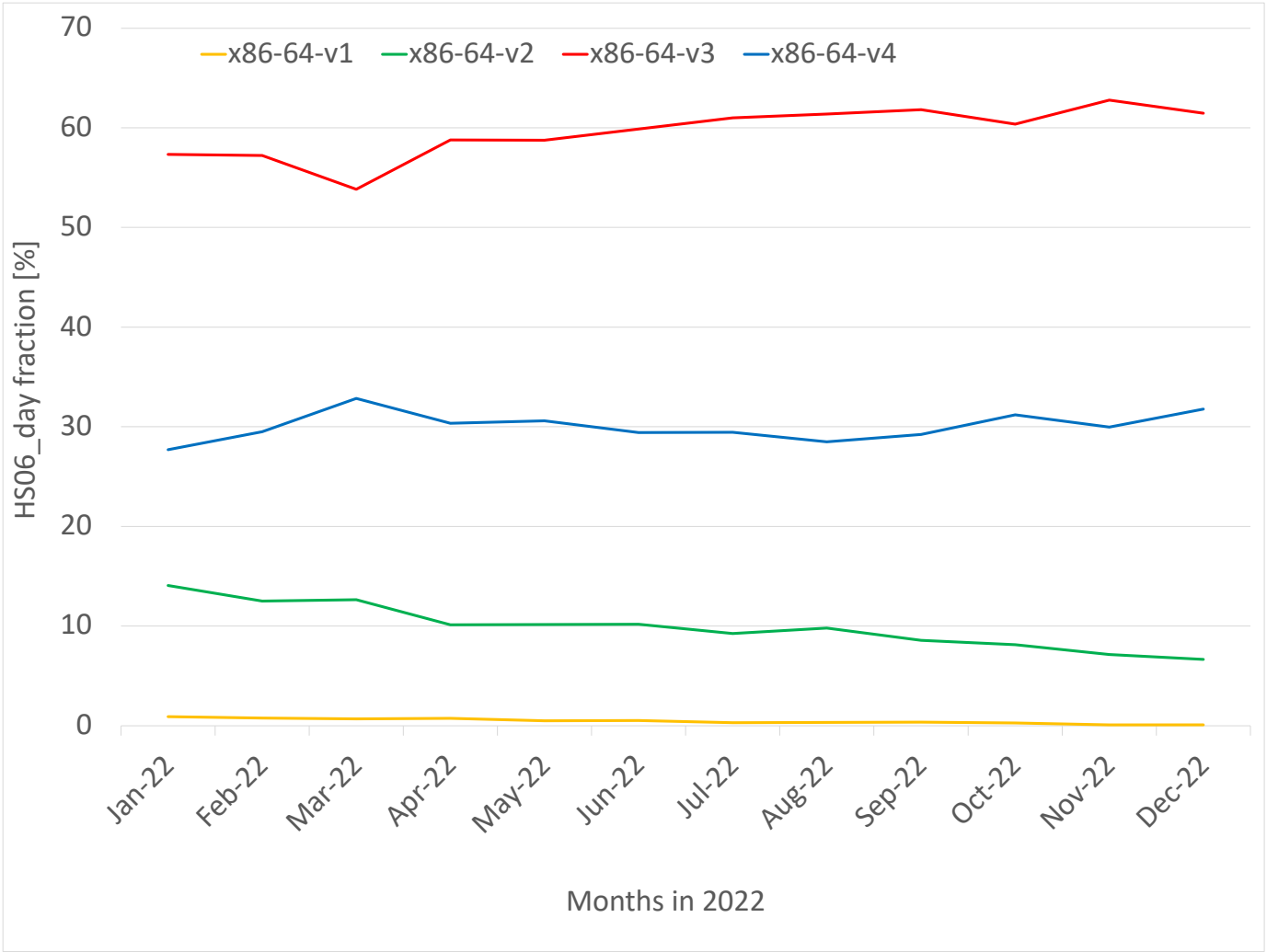
- Metric: $\sum(\mathrm{HS06\_day})$ $\Rightarrow$ sum of HEP-SPEC06 per day.
  HEP-SPEC06: the HEP-wide benchmark for measuring CPU performance.
  Duration: January-December 2022 (yearly). **[Numbers are in the backup]**



The order of dominance: x86-64-v3, x86-64-v4, x86-64-v2, x86-64-v1.

CPU popularity architecture-wise (all grid sites)

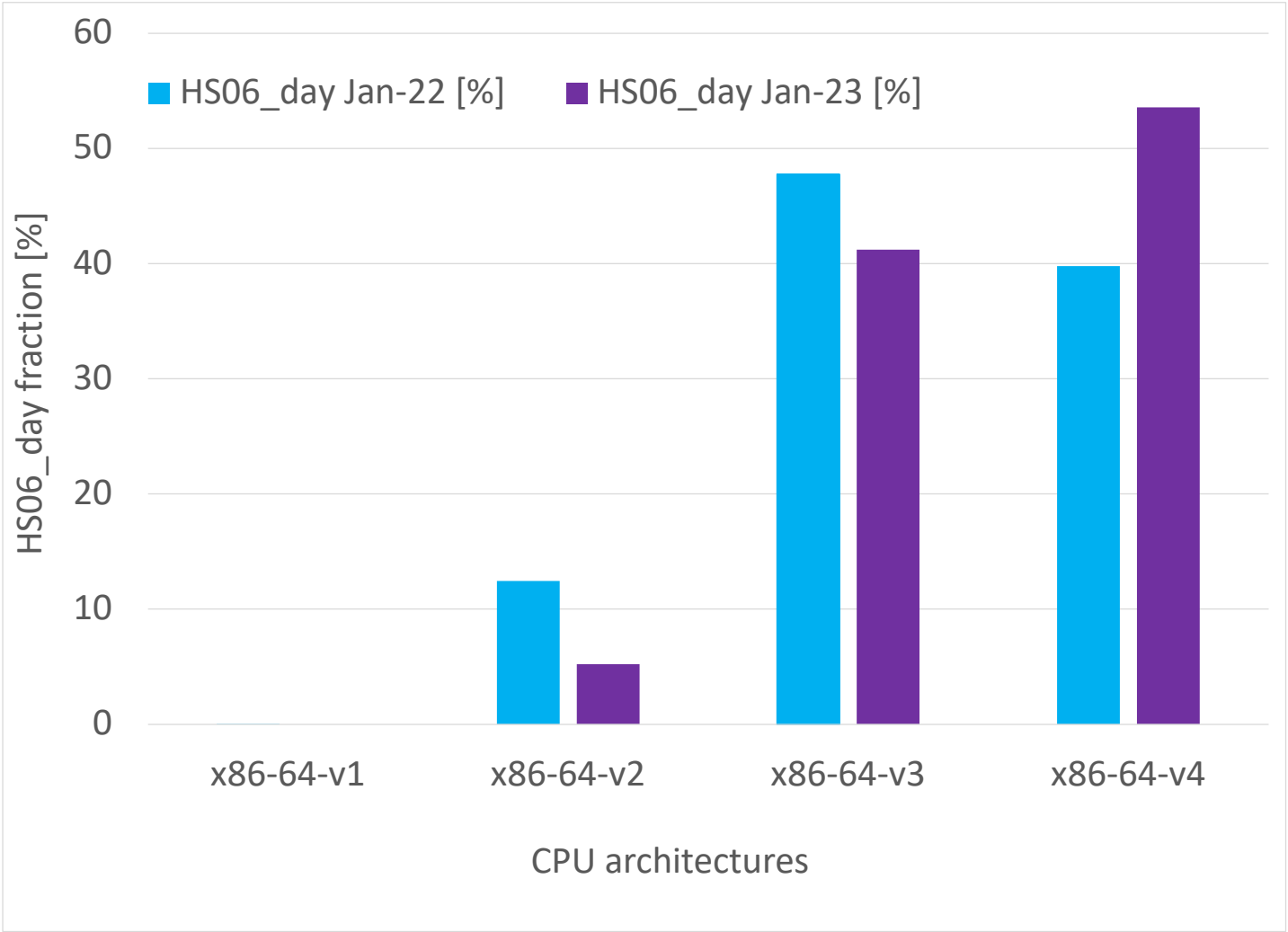
- Metric: $\sum(\mathrm{HS06\_day})$.
  Duration: January-December 2022 (monthly). **[Numbers are in the backup]**



A clear and steady trend of x86-64-v3 **dominance** followed by x86-64-v4, x86-64-v2, and x86-64-v1.

CPU popularity architecture-wise (UKI sites only)

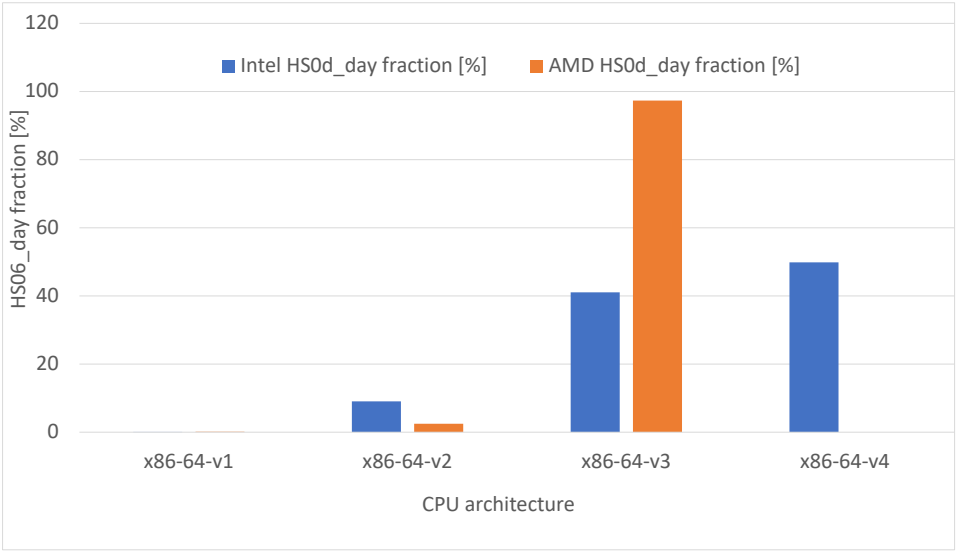
- Metric: $\sum(\mathrm{HS06\_day})$.
  Durations: January 2022 and January 2023. **[Numbers are in the backup]**

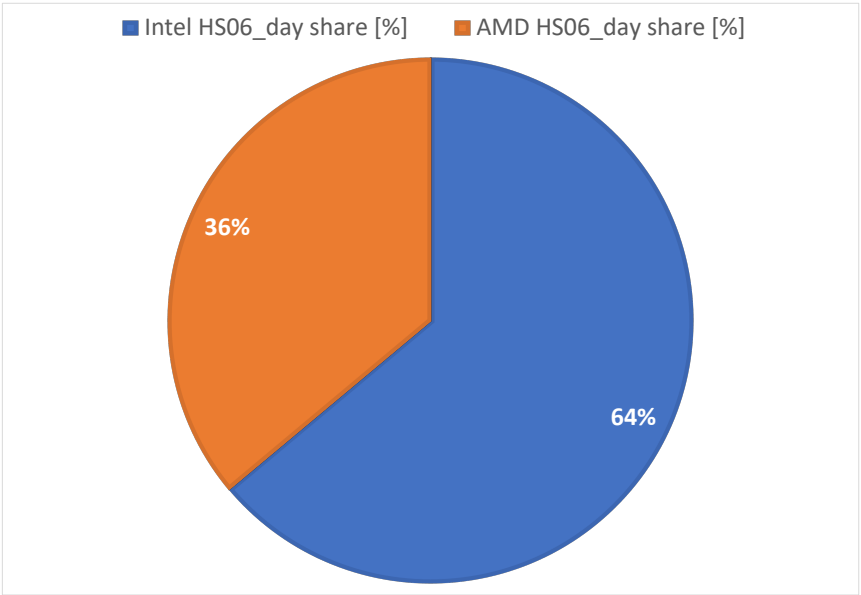


In January 2022, the dominant one is x86-64-v3, but **during January 2023, it's x86-64-v4**.

Intel and AMD CPUs (all grid sites)

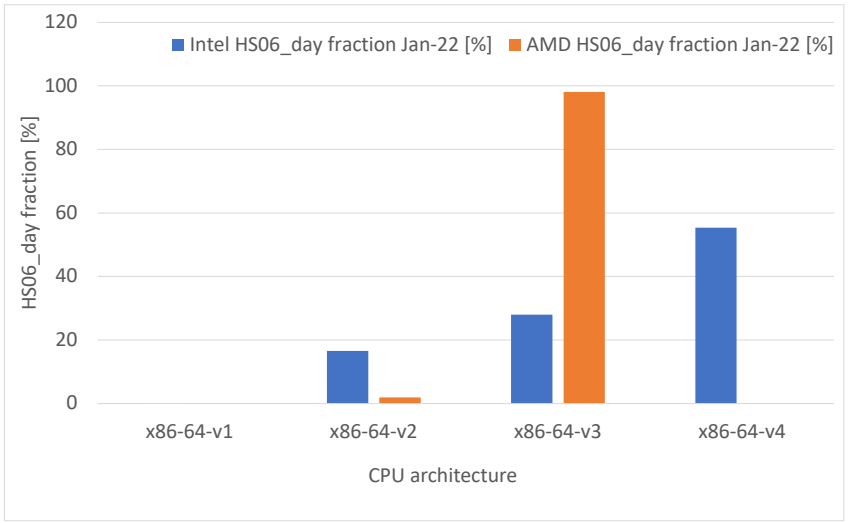
- Metric: $\sum(\mathrm{HS06\_day})$ [**top**] and $\sum(\mathrm{HS06\_day})$ share [**bottom**].
  Duration: January-December 2022 (yearly). **[Numbers are in the backup]**

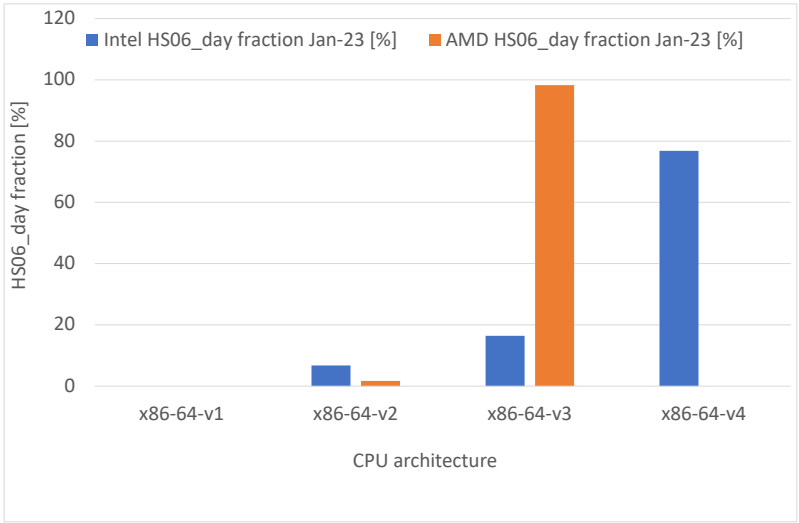


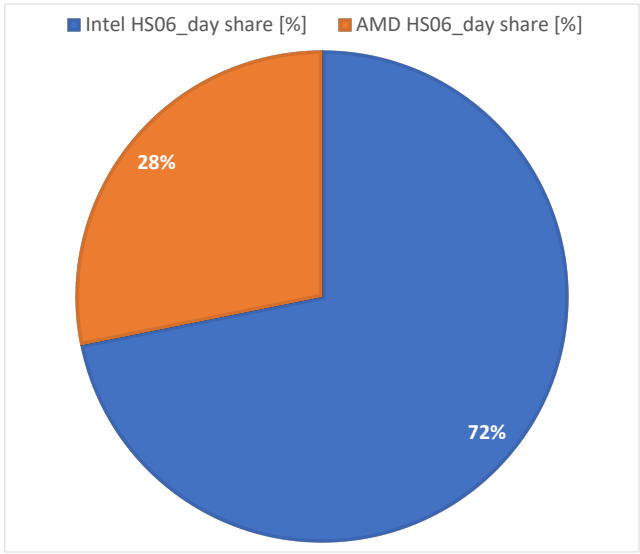


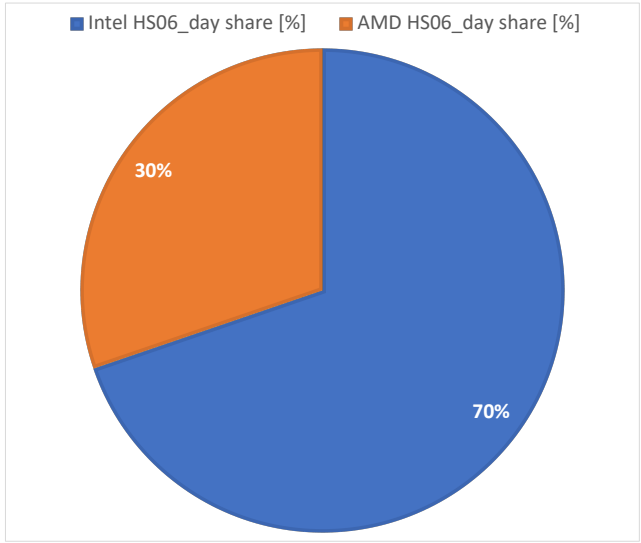
Intel and AMD CPUs (UKI sites only)

- Metric: $\sum(\mathrm{HS06\_day})$ [**top**] and $\sum(\mathrm{HS06\_day})$ share [**bottom**].
  Durations: January 2022 (**top/bottom-left**) and January 2023 (**top/bottom-right**).
  [Numbers are in the backup]



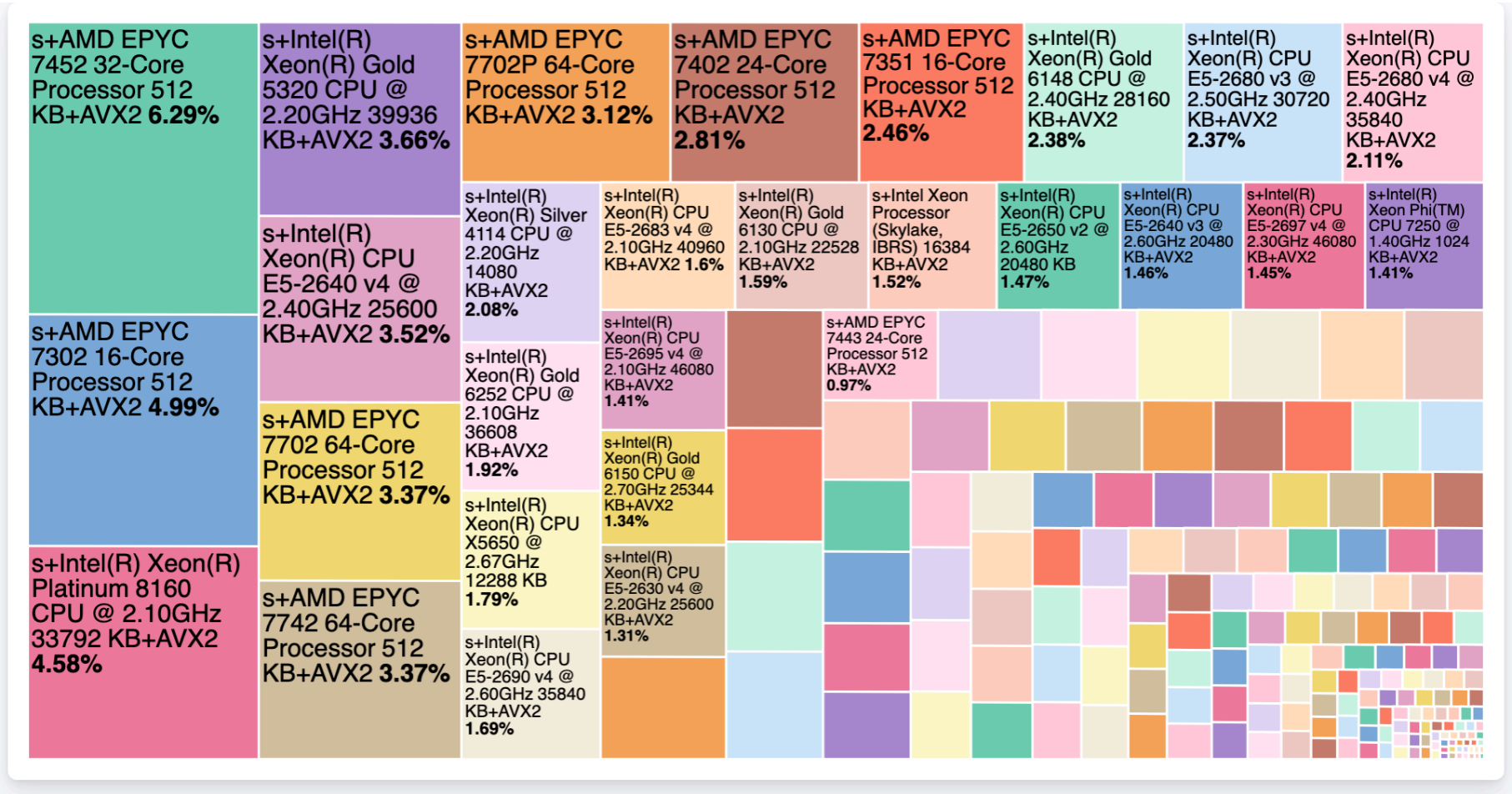






CPU popularity over one year (kibana *tree map*)

- Metric: $\sum(\mathrm{HS06\_day})$.
- Duration: January-December 2022 (yearly).
- **Search link**; top 200 CPUs.



AMD usage is concentrated over fewer different models.

Wrap-up

- On the grid, x86-64-v3 ($\sim$ 60%) is the most popular CPU architecture, followed by x86-64-v4 ($\sim$ 30%).
  - x86-64-v1's presence is negligible ($< 0.5\%$).
  - x86-64-v2 ($\sim$ 10%) is somewhere in the middle!
- UKI sites seem to be shifting towards x86-64-v4 (starting 2023)!
- Among the Intel CPUs, x86-64-v4 is beginning to dominate.
- $>$ **90%** of AMD CPUs are x86-64-v3.
  - Had there been a list of flags from AMD corresponding to x86-64-v4, it would be conducive in re-defining the architectures.
- HS06_day share is Intel-dominated $\sim \frac{2}{3}^{\mathbf{rd}}$.
- Ways to improve this study further:
  - Individual site-wise analysis.
  - National grid-wise analysis.
  - A catalogue of different Intel and AMD models.
- **Record of architectures and flags will be obtainable for all future jobs on kibana!**

Talking points

- What happens if a certain architecture, *e.g.*, x86-64-v1 is completely dropped from the grid?
- Is it viable to have architecture-specific sites? *E.g.*, X site has at least x86-64-v3 CPUs.
- GPUs on the grid may face similar challenges.
- Is there a way to match types of jobs to particular architectures?
- Next steps:
  - Investigation on non-x86-64 architectures, *e.g.*, ARM CPUs.
  - The Python script will eventually be made available on CernVM-File System.
- **There is enough data to discuss and propose the architectures ATLAS could require on the grid.**

Backup

CPU popularity architecture-wise (all grid sites)

All grid sites; January-December 2022 (yearly)	
CPU architecture	HS06_day fraction [%]
x86-64-v1	0.44
x86-64-v2	9.76
x86-64-v3	59.77
x86-64-v4	30.03

CPU popularity architecture-wise (all grid sites)

All grid sites; January-December 2022 (monthly); HS06_day fraction [%]				
Month-Year	x86-64-v1	x86-64-v2	x86-64-v3	x86-64-v4
January-2022	0.90	14.06	57.34	27.70
February-2022	0.77	12.50	57.23	29.50
March-2022	0.70	12.65	53.82	32.83
April-2022	0.74	10.14	58.78	30.34
May-2022	0.49	10.15	58.75	30.61
June-2022	0.53	10.18	59.88	29.41
July-2022	0.30	9.26	61.00	29.44
August-2022	0.32	9.81	61.38	28.49
September-2022	0.37	8.58	61.82	29.23
October-2022	0.29	8.14	60.37	31.20
November-2022	0.10	7.14	62.79	29.97
December-2022	0.09	6.66	61.47	31.78

CPU popularity architecture-wise (UKI sites only)

UKI sites; the same month after one year; HS06_day fraction [%]		
CPU architecture	January 2022	January 2023
x86-64-v1	0.04	0.0
x86-64-v2	12.44	5.23
x86-64-v3	47.75	41.20
x86-64-v4	39.77	53.57

Intel and AMD CPUs (all grid sites)

All grid sites; January-December 2022; HS06_day fraction [%]		
CPU architecture	Intel	AMD
x86-64-v1	0.06	0.14
x86-64-v2	9.03	2.50
x86-64-v3	41.03	97.36
x86-64-v4	49.88	0.0

Intel and AMD CPUs (UKI sites only)

UKI sites; January 2022; HS06_day fraction [%]		
CPU architecture	Intel	AMD
x86-64-v1	0.05	0.0
x86-64-v2	16.58	1.90
x86-64-v3	28.00	98.10
x86-64-v4	55.37	0.0

UKI sites; January 2023; HS06_day fraction [%]		
CPU architecture	Intel	AMD
x86-64-v1	0.0	0.0
x86-64-v2	6.74	1.73
x86-64-v3	16.46	98.27
x86-64-v4	76.80	0.0